HUSKY™

HUSKY HUNTER 2

User's Guide

Preface

The **Husky Hunter 2** microcomputer is a self contained and truly portable system. Its built-in operating system contains an accurate emulation of the world-standard CP/M operating system and virtual floppy disk drive, implemented in RAM memory Other built-in features include a powerful BASIC interpreter, a terminal emulation package, a flexible text editor with full-screen facilities based on the popular Wordstar and a comprehensive range of communications protocols. User memory is shared between work areas, program and data files.

The **Husky Hunter** has been an established product for several years, finding applications in many varied fields of use. During this period enhancements have been made to both the hardware and the software. For example, the physical screen size has been increased and the operating system has evolved from 9G, 9G06 and up to the present 9G08.

This manual refers to the present design of the **Husky Hunter 2**, but is equally applicable to previous Hunter units, except where differences are denoted.

(c) Copyright 1990 Husky Computers Limited
All Rights Reserved.

The information contained in this document is copyright and may not be reproduced, stored in a retrieval system in any form or by any means, electronic, mechanical, photocopying or otherwise without prior permission from the copyright holder, Husky Computers Limited.

Whilst every precaution has been taken in the preparation of this document, the publisher assumes no responsibility for errors and omissions neither is any liability assumed for damage resulting from the use of the information contained within this document.

Husky Computers Limited,
PO Box 135,
345 Foleshill Road,
Coventry CV6 5RW
England.

Telephone: Coventry (0203) 668181
Telex: 317450 Husky G
Facsimile: Coventry (0203) 680612

The following trademarks are the property of Husky Computers Limited.

Husky
Husky Hunter
Husky Hunter 2
Husky Reporter
Husky Oracle GT

Throughout this manual use of the term **Hunter 2** refers to **Husky Hunter 2** which is a trademark, the property of Husky Computers Limited.

CP/M is a registered trademark of Digital Research Corp.
WordStar is a trademark of MicroPro International Corp.
SuperCalc is a registered trademark of Digital Research Corp.
IBM, **2780** and **IBM-PC** are registered trademarks of IBM Corp.
Apple is a registered trademark of Apple Computer Inc.
MBASIC and **MS-DOS** are trademarks of Microsoft Inc.
dBase II is a trademark of Ashton-Tate Inc.
Turbo Pascal is a trademark of Borland International Inc.
AZTEC C is a trademark of Manx Software Systems.

Any other trademarks used in this guide are acknowledged.

Second Edition by Ed Phipps Documentation Services, Bognor Regis, England.

Second Edition (February 1990)

HUSKY HUNTER 2™

User's Guide

Contents

Chapter 1 Introduction to Hunter 2 1

1.1 Welcome to Hunter 2 1
1.2 Features 2
1.3 About this Manual 3
1.3.1 Manual's Purpose 3
1.3.2 How to Use this Manual 3
1.3.3 Manual Structure 3
1.3.4 Conventions 4

Chapter 2 Using Hunter 2 5

2.1 Introduction 5
2.2 Switching On 6
2.2.1 Display Needs Adjustment 6
2.2.2 Autostart Program Loaded 6
2.3 Selecting the File Manager 7
2.4 The System Files 9
2.4.1 DIR - File Directory 9
2.4.2 STAT - File Status 9
2.4.3 BAS - BASIC Interpreter 9
2.4.4 COMS - Initialise Communications 9
2.4.5 CLCK - Initialise Clock 9
2.4.6 TERM - Terminal Emulation 9
2.4.7 EDIT - Text Editor 9
2.4.8 KEYS - Alternate Functions Keys 10
2.5 The Keyboard 11
2.5.1 Environment 11
2.5.2 PWR and HLP 11
2.5.3 Turning on and off 11
2.5.4 ENTER 11
2.5.5 SHIFT 12
2.5.6 Cursor Keys 12
2.5.7 DEL, BS and Backarrow 12
2.5.8 CTL 12
2.5.9 FN and Function Keys 12
2.5.10 ESC and BRK 13
2.5.11 INS and LBL 13
2.6 The Screen 14
2.6.1 Introduction 14
2.6.2 Screen Modes 14

2.6.3 Screen Control 15
2.6.4 Virtual Screen 15
2.6.5 Screen Backlight 18
2.7 Communications 18
2.8 Batteries 20
2.8.1 Battery Installation 20
2.8.2 Main Battery 20
2.8.3 Low Battery Warning 21
2.8.4 Power Save 22
2.8.5 Battery Charging 22
2.8.6 Continuous Connection 23
2.8.7 Auto Timeout 23

Chapter 3 Troubleshooting 24

3.1 Introduction 24
3.2 Causes of Crashes 25
3.2.1 Illegal System Calls 25
3.2.2 Invalid User Machine Code 25
3.2.3 Physical Degradation 25
3.2.4 Operation Outside Specified Temperature Range 25
3.3 Symptoms of Crashes 26
3.3.1 Keyboard Lock-out 26
3.3.2 Clock Wipe-out 26
3.3.3 Program Corruption 26
3.3.4 Communications Parameter Corruption 26
3.4 Recovery 27
3.4.1 Keyboard Lock-out 27
3.4.2 Program Corruption 27
3.4.3 System Lock-out 27
3.4.4 File Corruption 27

Chapter 4 DEMOS File Manager 29

4.1 Introduction 29
4.2 Selecting DEMOS 29
4.3 Using DEMOS 31
4.3.1 Typing Commands 31
4.3.2 Command Interpretation 31
4.3.3 Executing CP/M programs 33
4.3.4 Executing BASIC programs 33
4.3.5 Continuous Execution 33
4.3.6 DEMOS Commands 34
4.4 Filing 35

4.4.1 Introduction 35
4.4.2 Filenames 35
4.4.3 Wildcards 35
4.4.4 Data Integrity 36
4.5 The DEMOS Commands 37
4.5.1 BAS 37
4.5.2 CLCK 38
4.5.3 COMS 40
4.5.4 CONT 42
4.5.5 DIR 43
4.5.6 EDIT 44
4.5.7 ERA 45
4.5.8 EXECUTE 46
4.5.9 FORMAT 47
4.5.10 INP 48
4.5.11 KEYS 50
4.5.12 LOAD 51
4.5.13 REN 52
4.5.14 SAVE 53
4.5.15 SEND 54
4.5.16 STAT 55
4.5.17 TERM 56
4.5.18 TYPE 57
4.5.19 /DO 58
4.6 DEMOS Error Messages 59

Chapter 5 Editor 61

5.1 Introduction 61
5.2 Selecting the Editor 61
5.2.1 Entry from the File Manager 61
5.2.2 Entry from BASIC 63
5.2.3 Automatic Cont Edit 64
5.2.4 File Size Limit 64
5.3 Editor Operation 65
5.3.1 Introduction 65
5.3.2 BASIC Files 65
5.3.3 The Keyboard 65
5.3.4 Line Length 67
5.4 Editor Commands 67
5.4.1 Introduction 67
5.4.2 ARROWS 68
5.4.3 CAPS LOCK & TAB 69

5.4.4 CHARACTER INSERT 70
5.4.5 DELETE 71
5.4.6 EXIT 72
5.4.7 FILE START AND END 73
5.4.8 FIND 74
5.4.9 LINE DELETE 75
5.4.10 LINE START AND END 76
5.4.11 PAGE SCROLL 77
5.4.12 SAVE 78
5.4.13 WORD SKIP 79
5.5 Editor Error Messages **80**
5.5.1 Beep 80
5.5.2 ** Failed 80
5.5.3 ** Wait 80
5.5.4 Disk full File length = *n* blocks 80
5.5.5 File R/O File length = *n* blocks 80
5.5.6 Memory overflow! 81

Chapter 6 Introduction to Communications 82

6.1 Introduction **82**
6.1.1 Background Information 82
6.2 Connecting to the Serial Port **85**
6.3 Configuring the Port **86**
6.3.1 Introduction 86
6.3.2 Using the On-screen Menu 86
6.3.3 Configuring the Port from Programs 87
6.4 The Port Set-up Options **87**
6.4.1 BAUD RATE 88
6.4.2 CTS 88
6.4.3 DCD 89
6.4.4 DSR 89
6.4.5 DTR 89
6.4.6 ECHO 90
6.4.7 LF 90
6.4.8 NULL 91
6.4.9 PRTCL (Protocol) 91
6.4.10 PTY (Parity) 92
6.4.11 RTS 93
6.4.12 RX T/O (Receive Timeout) 94
6.4.13 SERIG 94
6.4.14 TX T/O (Transmit Timeout) 95

6.5 Handshaking **96**
6.6 Asynchronous Protocols **97**
6.6.1 Introduction 97
6.6.2 NONE 97
6.6.3 XON/XOFF (DC1/DC3) 97
6.6.4 ETX/ACK 98
6.6.5 ACK/NAK 99
6.6.6 SYSTIME 99
6.7 Synchronous Protocols **102**
6.7.1 IBM 2780/3780 on **Hunter 2** 102
6.7.2 Synchronous Communication 103
6.7.3 Binary Synchronous Communications (BSC) 105
6.7.4 Link Control Characters 106
6.7.5 Transparency 108
6.7.6 Message Transmission 111
6.7 7 **Hunter 2** 2780 Implementation Features 115
6.7.8 Configuring the Protocol (SETSYNC8) 120
6.7.9 Use In Terminal Emulation Mode 126
6.7.10 Hardware Handshaking 126
6.7.11 Hardware Configuration 127
6.7.12 **Hunter 2** to **Hunter 2** Connection 127
6.8 Communications Setup Error Messages **128**

Chapter 7 Using Communications 129

7.1 Introduction **129**
7.1.1 Using Communications 129
7.1.2 Standard applications 129
7.1.3 Programming 130
7.1.4 Starting communication 130
7.2 Transferring Files **131**
7.2.1 File Description 131
7.2.2 8-bit Transfer 132
7.2.3 Intel Hex Format Transfer 134
7.3 Using Hunter 2 as a Terminal **137**
7.3.1 Introduction 137
7.3.2 Connecting to the Host 137
7.3.3 Selecting Terminal Emulation 137
7.3.4 Protocols 138
7.3.5 Buffering 138
7.3.6 Auto Power Off 138
7.4 Advanced Communications Techniques **139**
7.4.1 Introduction 139

7.4.2 9600 Baud Communications 139
7.4.3 Using Bisynchronous Protocols 141
7.4.4 Using the SYSTIME Protocol 145
7.5 Communications Errors **147**

Chapter 8 Looking After Hunter 2 149

8.1 Everyday Use **149**
8.2 Case Sealing **149**
8.3 Humidity Indicator **150**
8.4 Storage **150**

Chapter 9 Accessories 151

9.1 Introduction **151**
9.2 Barcodes and Light Pens **152**
9.2.1 Introduction 152
9.2.2 Using a Barcode Wand 152
9.3 Battery Charger **154**
9.3.1 Description 154
9.3.2 WARNING 154
9.3.3 Operation 154
9.3.4 Replacing the Rechargeable Batteries 155
9.3.5 Mains Operation 155
9.4 Husky Oracle GT Disk Drive **156**
9.4.1 Operation of the **Oracle GT** 156
9.4.2 Setting up 160
9.4.3 Using the Oracle GT with **Hunter 2** 165

Chapter 10 Introduction to Hunter BASIC 171

10.1 Introduction **171**
10.1.1 Selecting **Hunter** BASIC 171
10.1.2 BASIC Programs 172
10.1.3 Autostart BASIC Programs 172
10.2 BASIC Syntax **173**
10.2.1 Types of Keyword 173
10.2.2 Expressions and Operators 173
10.2.3 Operator Precedence 178
10.3 Program Lines **180**
10.3.1 Line Numbers 180
10.3.2 Multi-statement Lines 180
10.4 Types of Variables **181**
10.4.1 Numeric Variables 181
10.4.2 Numeric Arrays 181
10.4.3 String Variables 182
10.5 Variable Structure **183**
10.5.1 Simple Numeric Variables 183
10.5.2 Numeric Array Variables 183
10.5.3 Double Precision Array Variables 183
10.5.4 String Variables 184
10.6 Editor **185**
10.7 Keyboard **186**
10.7.1 Introduction 186
10.7.2 Redefining the Keyboard 186
10.7.3 Special Key Codes 187
10.7.4 Function Keys 187
10.8 Program Limits **188**
10.8.1 Ranges 188
10.8.2 Precision 188
10.9 Memory overhead **189**
10.9.1 Program storage 189
10.9.2 Dynamic (Run-time) Memory Allocation 189
10.10 Graphics **190**
10.10.1 Introduction 190
10.10.2 Graphics Demonstrations 191
10.11 Using Machine Code **192**
10.11.1 The Microprocessor 192
10.11.2 Calling a Machine Code Routine 192
10.11.3 Using RAM 194
10.11.4 Converting Machine Code to POKE Statements 198
10.12 Data Storage Arrays **199**
10.12.1 Types of Arrays 199
10.12.2 **Simple Variables** 199
10.12.3 Double Precision Arrays 199
10.12.4 String Arrays 199
10.12.5 Array Structures 199
10.12.6 Creating Arrays 200
10.12.7 Accessing Array Elements 201
10.12.8 Searching Arrays 201
10.12.9 Using **Hunter 2**'s Screen 201
10.12.10 Help Statement 203
10.13 Increasing Battery Life **205**
10.14 Low Power Warning **206**
10.15 Power Handling Commands **208**
10.15.1 Introduction 208
10.15.2 Default Conditions 208

Husky Hunter 2 User's Guide: Contents

10.15.3 POWER CONT 208
10.15.4 POWER n 208
10.15.5 POWER OFF 208
10.15.6 POWER OFF, RESUME 209
10.15.7 ON POWER GOTO 209
10.15.8 ON POWER RESUME 209
10.15.9 Timeouts 209
10.16 File Handling **210**
10.16.1 Introduction 210
10.16.2 File Numbers 210
10.16.3 File Handling Commands 210
10.16.4 Determining Disk Space 212
10.17 Random Access Filing **213**
10.17.1 Stage 1 - Installing Machine Code 213
10.17.2 Stage 2 - File Initialisation 214
10.17.3 Stage 3 - Opening the Files for Use 215
10.17.4 Stage 4 - Transferring Data to and from Files 216
10.17.5 Stage 5 - Closing Files in Machine Code and BASIC 218
10.17.6 Demonstration Program 219
10.18 Executing Programs from the RAM Disk **222**
10.18.1 Executing a Single Program 222
10.18.2 Program Switching between the TPA and the RAM Disk 222
10.18.3 Program Switching Multiple Programs in situ within the RAM Disk 224
10.18.4 Chaining Programs 231
10.19 'Alarm Clock' Feature **233**
10.20 Errors and Warnings **234**

Chapter 11 Reference to Hunter BASIC 236

11.1 ABS **237**
11.1.1 Return absolute value of the argument 237
11.2 ARG **238**
11.2.1 Pass parameters to machine code subroutine 238
11.3 ASC **239**
11.3.1 Return decimal ASCII code of string 239
11.4 ATN **240**
11.4.1 Return Arc-tangent 240
11.5 BEEP **241**
11.5.1 Generate a beep 241
11.6 CALL **242**
11.6.1 Call a machine code subroutine 242

11.7 CHAR **243**
11.7.1 Select character set 243
11.8 CHR$ **244**
11.8.1 Return character corresponding to ASCII code 244
11.9 CIRCLE **245**
11.9.1 Draw a circle 245
11.10 CLEAR **246**
11.10.1 Set all variables to zero. 246
11.11 CLOSE **247**
11.11.1 Close files 247
11.12 CLS **248**
11.12.1 Clear display screen 248
11.13 COM **249**
11.13.1 Control communications interrupt handling 249
11.14 CONT **250**
11.14.1 Continue execution after STOP 250
11.15 COS **251**
11.15.1 Return the Cosine of the argument 251
11.16 CRT **252**
11.16.1 Use serial port as keyboard and screen 252
11.17 CUROFF/CURON **253**
11.17.1 Turn cursor off and on 253
11.18 DATA **254**
11.18.1 Store data for use by READ 254
11.19 DATE$ **255**
11.19.1 Return or set the date 255
11.20 DAY$ **256**
11.20.1 Return or set the day 256
11.21 DEFSEG **257**
11.21.1 Define RAM page for machine code access 257
11.22 DIM **258**
11.22.1 Set up arrays 258
11.23 EDIT **260**
11.23.1 Invoke the **Hunter 2** text editor 260
11.24 END **261**
11.24.1 Terminate user program execution 261
11.25 EOF **262**
11.25.1 Check for end of file 262
11.26 ERR and ERL **263**
11.26.1 Return error code and line number 263
11.27 ERROR **264**
11.27.1 Simulate a BASIC error 264
11.28 EXP **265**

11.28.1 Return the value of e raised to the power of X 265
11.29 FILES **266**
11.29.1 List user files 266
11.30 FIX **267**
11.30.1 Return integer version of argument 267
11.31 FOR **268**
11.31.1 Repeat a sequence of instructions a number of times 268
11.32 FRE **270**
11.32.1 Return the number of bytes free for BASIC 270
11.33 GOSUB **271**
11.33.1 Jump to a BASIC subroutine 271
11.34 GOTO **272**
11.34.1 Branch unconditionally to a specified line 272
11.35 HELP **274**
11.35.1 Define location of help text 274
11.36 IF **275**
11.36.1 Execute statements depending on an expression or comparison 275
11.37 INCHR **277**
11.37.1 Input a single character from the keyboard 277
11.38 INKEY **278**
11.38.1 Scan the keyboard and return an ASCII code 278
11.39 INKEY$ **279**
11.39.1 Scan the keyboard and return a character 279
11.40 INP **280**
11.40.1 Input a byte from a port 280
11.41 INPUT **281**
11.41.1 Input a line of data from the keyboard 281
11.42 INPUT USING **283**
11.42.1 Input and validate a line of data from the keyboard 283
11.43 INPUT# **286**
11.43.1 Input data from a file 286
11.44 INSTR **287**
11.44.1 Search for substring within a string 287
11.45 INT **288**
11.45.1 Round argument down to the next lower integer 288
11.46 JSR$ **289**
11.46.1 Create right justified string from numeric 289
11.47 KEY **290**
11.47.1 Control function keys 290
11.48 KEY(n) **292**
11.48.1 Control trapping of function keys 292
11.49 KILL **294**

11.49.1 Delete a file 294
11.50 LEFT$ **295**
11.50.1 Return the first n characters of a string 295
11.51 LEN **296**
11.51.1 Return the length of a string 296
11.52 LET **297**
11.52.1 Assign a value to a variable 297
11.53 LINCHR **299**
11.53.1 Input a single character from the RS232 port 299
11.54 LINE **300**
11.54.1 Draw lines and boxes on the screen 300
11.55 LINPUT **301**
11.55.1 Input a line of data from the RS232 port 301
11.56 LIST **302**
11.56.1 List program on the screen 302
11.57 LLIST **303**
11.57.1 List program to the RS232 port 303
11.58 LLOAD **304**
11.58.1 Load a BASIC program from the RS232 port 304
11.59 LN **306**
11.59.1 Return a natural logarithm 306
11.60 LOAD **307**
11.60.1 Load a BASIC program from the RAM disk 307
11.61 LOC **308**
11.61.1 Return the current position in a sequential file 308
11.62 LOCATE **309**
11.62.1 Position cursor on the screen 309
11.63 LOG **310**
11.63.1 Generate logarithm to base 10 310
11.64 LOPCHR **311**
11.64.1 Output characters to the RS232 port 311
11.65 LPRINT **312**
11.65.1 Output text to the RS232 port 312
11.66 LTRON **313**
11.66.1 Trace execution by sending line numbers to RS232 port 313
11.67 MAXFILES **314**
11.67.1 Files open limit 314
11.68 MID$ **315**
11.68.1 Return or assign a substring 315
11.69 NAME **316**
11.69.1 Rename a file 316
11.70 NEW **317**

11.70.1 Clear the current program, variables, etc 317
11.71 NEXT **318**
11.71.1 Complete a FOR..NEXT loop 318
11.72 ON BREAK **319**
11.72.1 Control trapping of the Break key 319
11.73 ON COM **320**
11.73.1 Define a subroutine to execute when serial data is received 320
11.74 ON COMMS **321**
11.74.1 Control comms error handling 321
11.75 ON ERROR **322**
11.75.1 Control trapping of BASIC errors 322
11.76 ON n GOSUB **323**
11.76.1 Call one of a list of subroutines depending on *n* 323
11.77 ON n GOTO **324**
11.77.1 Branch to one of a list of lines depending on *n* 324
11.78 ON KEY **325**
11.78.1 Define a soft key handling routine 325
11.79 ON POWER **326**
11.79.1 Control trapping of the Power key (PWR) 326
11.80 ON POWER RESUME **327**
11.80.1 Put **Hunter 2** into CONT mode 327
11.81 ON TIME$ **328**
11.81.1 Define a routine to execute at a specified time 328
11.82 OPCHR **329**
11.82.1 Output characters to the screen 329
11.83 OPEN **330**
11.83.1 Open a file for sequential input/output 330
11.84 OUT **332**
11.84.1 Output a byte to a port 332
11.85 PEEK **333**
11.85.1 Read a byte from memory 333
11.86 PI **334**
11.86.1 Return the value of PI 334
11.87 POINT **335**
11.87.1 Return the colour of a screen pixel 335
11.88 POKE **336**
11.88.1 Write bytes to memory 336
11.89 POP **337**
11.89.1 Fetch values from the machine code stack 337
11.90 POS **338**
11.90.1 Return the horizontal position of the cursor 338
11.91 POWER **339**

11.91.1 Set auto power-off period 339
11.92 POWER CONT **340**
11.92.1 Disable the power key and power off time-out 340
11.93 POWER OFF **341**
11.93.1 Switch off **Hunter 2** 341
11.94 POWER OFF, RESUME **342**
11.94.1 Switch off **Hunter 2** with automatic continuation 342
11.95 PRINT **343**
11.95.1 Display data on the screen 343
11.96 PRINT# **345**
11.96.1 Write data to a file 345
11.97 PSET and PRESET **347**
11.97.1 Set or reset screen pixels 347
11.98 PUSH **348**
11.98.1 Push values onto the machine code stack 348
11.99 READ **349**
11.99.1 Read constants or variables from a DATA statement 349
11.100 REM **350**
11.100.1 Ignore remainder of line 350
11.101 RESTORE **351**
11.101.1 Move data pointer used by READ 351
11.102 RESUME **352**
11.102.1 Return from an error-handling routine 352
11.103 RETURN **353**
11.103.1 Return from a BASIC subroutine 353
11.104 RIGHT$ **354**
11.104.1 Return the last n characters of a string 354
11.105 RND **355**
11.105.1 Return a random number 355
11.106 RUN **356**
11.106.1 Run a BASIC program 356
11.107 SAVE **357**
11.107.1 Save current program as a file 357
11.108 SCREEN **358**
11.108.1 Select text or graphics screen 358
11.109 SGN **359**
11.109.1 Return a value indicating the sign of the argument 359
11.110 SIN **360**
11.110.1 Return Sine of an angle 360
11.111 SOUND **361**
11.111.1 Generate a tone 361
11.112 SPACE$ **362**
11.112.1 Return a string of spaces 362

11.113 SPC 363
11.113.1 Print spaces in a PRINT statement 363
11.114 SQR 364
11.114.1 Return square root 364
11.115 SRCH 365
11.115.1 Search an array 365
11.116 STOP 367
11.116.1 Stop execution with a message 367
11.117 STR$ 368
11.117.1 Return a string representation of a numeric 368
11.118 STRING$ 369
11.118.1 Return a string of a specified character 369
11.119 SWAP 370
11.119.1 Exchange the values of two variables 370
11.120 SYSTEM 371
11.120.1 Return to File Manager 371
11.121 TAB 372
11.121.1 Move the cursor to a specified column 372
11.122 TAN 373
11.122.1 Return tangent of an angle 373
11.123 TIME$ 374
11.123.1 Return or set the time 374
11.124 TRON and TROFF 375
11.124.1 Control program tracing 375
11.125 VAL 376
11.125.1 Return numeric version of a string 376
11.126 VARPTR 377
11.126.1 Return address of a variable 377
11.127 WAND 378
11.127.1 Select barcode protocol 378
11.128 WHILE..WEND 379
11.128.1 Execute statements while a condition is true 379
11.129 WINCHR 381
11.129.1 Input a single character from a barcode wand 381
11.130 WINPUT 382
11.130.1 Input a line from a barcode wand 382
11.131 WRITE# 383
11.131.1 Output data to a file 383

Chapter 12 Developing Software for the Hunter 2 385

12.1 Introduction 385

12.2 Interpreted Microsoft BASIC 385
12.3 Compiled Microsoft BASIC 386
12.3.1 Development on **Hunter 2** 386
12.3.2 Development on a Host Computer 387
12.4 Turbo Pascal 3 387
12.5 C Compilers 388
12.6 Cross Compilers 389
12.7 Assemblers and Machine Code 391
12.8 Software Manufacturers 391
12.9 Availability 392

Chapter 13 Assembly Language Programming 393

13.1 Introduction 393
13.1.1 Memory Organisation 393
13.1.2 Paging from Machine Code 396
13.1.3 The Stack 396
13.1.4 The PWR Key 396
13.1.5 Cycle Time 396
13.1.6 CP/M Programming 396
13.1.7 File Organisation 397
13.1.8 Executing Programs 397
13.2 Introduction to CP/M Programming 398
13.2.1 System Calls Summary 398
13.2.2 File Control Block 399
13.3 Hunter 2 System Calls 399
13.3.1 System Reset (Call 0) 401
13.3.2 Console Input (Call 1) 401
13.3.3 Console Output (Call 2) 401
13.3.4 Reader Input (Call 3) 401
13.3.5 Punch Output (Call 4) 401
13.3.6 List Output (Call 5) 401
13.3.7 Direct Console I/O (Call 6) 402
13.3.8 Get I/O Byte (Call 7) 402
13.3.9 Set I/O Byte (Call 8) 402
13.3.10 Print String to Console (Call 9) 402
13.3.11 Read Console Buffer (Call 10) 403
13.3.12 Get Console Status (Call 11) 404
13.3.13 Return Version Number (Call 12) 404
13.3.14 Reset Disk System (Call 13) 404
13.3.15 Select Disk (Call 14) 404
13.3.16 Open File (Call 15) 405

13.3.17 Close File (Call 16) 405
13.3.18 Search for First (Call 17) 405
13.3.19 Search for Next (Call 18) 405
13.3.20 Delete File (Call 19) 406
13.3.21 Read Sequential (Call 20) 406
13.3.22 Write Sequential (Call 21) 406
13.3.23 Make File (Call 22) 406
13.3.24 Rename File (Call 23) 406
13.3.25 Return Login Vector (Call 24) 407
13.3.26 Return Current Disk (Call 25) 407
13.3.27 Set DMA Address (Call 26) 407
13.3.28 Get Allocation Address (Call 27) 407
13.3.29 Write Protect Disk (Call 28) 407
13.3.30 Get Read Only Vector (Call 29) 407
13.3.31 Set File Attributes (Call 30) 408
13.3.32 Get Disk Parameter Address (Call 31) 408
13.3.33 Get or Set User Code (Call 32) 408
13.3.34 Read Random (Call 33) 408
13.3.35 Write Random (Call 34) 408
13.3.36 Compute File Size (Call 35) 409
13.3.37 Set Random Record (Call 36) 409
13.3.38 Reset Drive (Call 37) 409
13.3.39 Set up Serial Port (Call 38) 409
13.3.40 Initialise Clock (Call 39) 409
13.3.41 Write Random with Zero Fill (Call 40) 410
13.3.42 Initialise Communications (Call 41) 410
13.3.43 Get Serial Status (Call 42) 410
13.3.44 Terminal Emulation Call (Call 43) 410
13.3.45 Wand Input Call (Call 44) 411
13.3.46 Hex Format Output (Call 45) 411
13.3.47 Hex Format Input (Call 46) 411
13.3.48 Fetch Key without Echo (Call 47) 411
13.3.49 Read Clock (Call 48) 412
13.3.50 Reserved (Call 49) 412
13.3.51 Force transmission (Call 50) 412
13.3.52 Transmit handshake (Call 51) 412
13.3.53 Reserved (Call 52) 412
13.3.54 Reserved (Call 53) 413
13.3.55 Serial output status (Call 54) 413
13.4 Memory Locations 414
13.5 I/O Address Map 421
13.6 Redefining the Keyboard 423
13.6.1 Special Codes 425

13.7 Controlling Power-Off 428
13.8 Power-up 430
13.9 Using EPROMs for Programs (9G08 only) 431
13.9.1 Introduction 431
13.9.2 EPROM Application 432
13.9.3 Description of EPROM Option 433
13.9.4 ROM Structure 433
13.9.5 Description of Operation 436
13.9.6 Extra DEMOS Commands 437
13.9.7 Machine Code Linkages 438
13.9.8 Important Note 441

Chapter 14 Using Escape Codes (9G08 only) 442

14.1 Introduction 442
14.2 LINE: draw lines or boxes 444
14.2.1 Remarks 444
14.3 CURON/CUROFF: turn cursor on/off 444
14.3.1 Remarks 444
14.4 POINT: read pixel colour 444
14.4.1 Remarks 445
14.5 LOCATE: move cursor and turn it on/off 445
14.5.1 Remarks 445
14.6 CHAR: character set and video attributes 446
14.7 SCREEN: select graphics or text mode 446
14.8 PSET: set a pixel to black 446
14.8.1 Remarks 446
14.9 PRESET: reset a pixel to white 446
14.9.1 Remarks 447
14.10 CIRCLE: draw a circle 447
14.10.1 Remarks 447
14.11 Caps Lock: set or reset caps lock mode 447
14.12 Set Viewing Angle 448
14.13 Backlight: turn backlight on or off 448
14.14 Set Security Escape Code 448
14.15 Default Security Escape Code: set escape code to default 448
14.15.1 Remarks 448
14.16 INS Key: restore window position 448
14.16.1 Remarks 449
14.17 Audible Output Control: turn sound output on or off 449

Chapter 15 Interfacing 451

15.1 RS232 Interface (Communications Port) 451
15.1.1 Introduction 451
15.1.2 Connector Layout 451
15.1.3 Pinouts 451
15.1.4 Electrical Characteristics 452
15.1.5 Interface Power Control 453
15.1.6 Connector Locking 453
15.2 Single Bit Input Port (LEMO) 454
15.2.1 Introduction 454
15.2.2 Circuit Implementation 454
15.2.3 The LEMO Connector 455

Chapter 16 Batteries 456

16.1 Introduction 456
16.2 Battery Capacity 456
16.3 Power Modes 456
16.4 Example Power Consumption 457
16.4.1 Each Visit 457
16.4.2 Battery Life 458
16.5 Data Storage Life 458
16.6 Mains (Line) Powered Installations 459
16.6.1 Permanent Charging 459
16.6.2 Mains Power Supply 461
16.6.3 Conclusions 461
16.6.4 Technical Details 461

Chapter 17 Hunter 2 Specification 463

17.1 Physical 463
17.2 Fascia 463
17.3 Programming 463
17.4 Communications 464
17.5 Graphics 464
17.6 Sound 464
17.7 Character Sets in Graphics Mode 464
17.8 Memory 465
17.9 Real Time Clock 465
17.10 Microprocessor 465
17.11 Batteries 465

Chapter 18 Operating Environment 466

18.1 Operating Temperature 466
18.2 Humidity 466
18.3 Waterproofing and Dustproofing 466
18.4 Storage Conditions 467
18.5 Electromagnetic Compatibility 467
18.6 Resistance to Shock, Vibration and Impact 467

Chapter 19 ASCII Character Set 469

Chapter 20 ASCII to EBCDIC Conversion Table 470

Chapter 21 Hexadecimal to Decimal Conversion Table 472

Chapter 22 Useful Utility Programs 473

22.1 Monitoring for low batteries in application programs 473
22.2 Restoring an accidentally deleted file 474
22.3 Identifying the operating system version from an application 475
22.4 Converting files to Intel Hex format 476
22.5 Using the Oracle GT from Hunter BASIC 477
22.5.1 Introduction 477
22.5.2 The ORACLE GT disk functions 478
22.5.3 Using the disk functions 481
22.5.4 Error handling 483
22.5.5 Use with machine code or compiled applications 485
22.6 Accessing the RTC from applications 486
22.6.1 Implementing the routines in compiled MBASIC 488
22.6.2 Implementing the routines in TURBO PASCAL 489
22.7 Using alternate character sets in text mode 490

Chapter 23 Intel Hex Format 492

Chapter 24 Four Octave Sound Range 494

Chapter 25 Graphics Routines 495

Husky Hunter 2 User's Guide: Contents

25.1 Introduction 495
25.2 Program Description 495
25.3 Graphics Functions 496
25.3.1 Clear the LCD Screen 496
25.3.2 Set LCD Screen to Graphics Mode 496
25.3.3 Set LCD Screen to Text Mode 496
25.3.4 Switch Cursor on 496
25.3.5 Switch Cursor off 496
25.3.6 Select a Graphics Character Set 497
25.3.7 Set/Reset a Pixel 497
25.3.8 Draw a Circle 497
25.4 Listing 498
25.5 Graphics Demonstration Program 501
25.6 Reading a Pixel's Colour 505

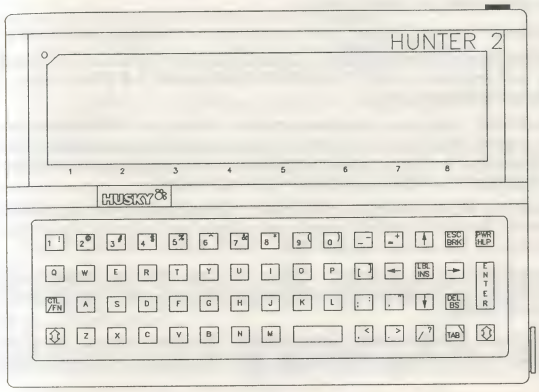
Chapter 26 IBM2780 Protocol Menu Program for 9G06 507
26.1 Introduction 507
26.2 SETSYNC Listing 507

HUSKY HUNTER 2™

Part 1
Introduction

Chapter 1 Introduction to Hunter 2

1.1 Welcome to Hunter 2



Husky **Hunter 2** is one of the smallest and toughest CP/M microcomputers ever made.

It is resistant to moisture, dust, vibration, shock and electromagnetic interference. But please don't abuse it. Inside the impervious cast aluminium case is some of the most advanced technology money can buy Don't drop **Hunter 2** needlessly, use it to strike other objects or pile heavy things on top of it.

If you can't find the information you need in this manual or you have any problems with **Hunter 2**, please call us. We've done all we can to make **Hunter 2** a practical and utilitarian tool, not a frustrating encumbrance. We hope you agree. After all, **Hunter 2** is our baby

1.2 Features

Measuring just 216 x 156 x 32 mm (8.5" x 6.2" x 1.3") and weighing approximately 1200g (3lb), **Hunter 2** is a completely self-contained computer system, sealed against moisture, dust and other hazards.

NEVER attempt to open the case - there are no user serviceable parts inside.

The CP/M emulating operating system, contains a flexible text editor, RAM-disk emulation and provides a totally compatible environment for a vast range of commercially available software.

Hunter 2's BASIC Interpreter has been specially developed to be compatible with other popular BASICs and to meet the extra demands of a portable system. Special statements, for example to filter input data so that only formats consistent with the program are accepted, simplify user programming.

Communications with instruments or computers, including main-frames, are user programmable. Protocols can be chosen from a wide selection, including a choice of synchronous or asynchronous types.

CP/M compatibility is an important feature. A large range of commercially available software, including packages such as spreadsheets, can run on **Hunter 2** with no modifications.

Disk emulation in RAM memory allows standard programs such as WordStar and SuperCalc to run without modification, while files can be exchanged easily

1.3 About this Manual

1.3.1 Manual's Purpose

This manual is intended to fulfil several functions:

* As an introduction to using and programming **Hunter 2**.
* As a source of technical data about **Hunter 2**.
* As a guide to achieving **Hunter 2**'s full potential in diverse user applications.

It is not intended as an introductory guide for a first time computer user, nor as a tutorial in programming. For guidance on these subjects, refer to any of the excellent introductory works dealing with computers, microcomputer BASIC and Z80 assembly language now available.

1.3.2 How to Use this Manual

This manual is split into 6 separate parts, as described below You should read the remainder of this part and then those parts that are appropriate to the way you are using **Hunter 2**.

Cross-references are kept to a minimum to make the text more readable; if you need further information on a topic, refer to the index at the end of the manual.

1.3.3 Manual Structure

The manual is split into a six parts, as follows:

1) INTRODUCTION: An introduction to **Hunter 2** and this manual.
2) FACILITIES: the standard **Hunter 2** facilities, including the File Manager, editor, communications, using CP/M software, looking after **Hunter 2** and the accessories available.
3) BASIC: programming using built-in BASIC language. **Hunter** BASIC is based on Microsoft BASIC, while providing a wide range of additional facilities suited to programming for portable applications.
4) CP/M and ASSEMBLY LANGUAGE: programming the **Hunter 2**'s Z80-compatible microprocessor directly, for maximum versatility and control over processing. **Hunter 2** provides a CP/M-compatible

environment to allow third party applications to run, while also providing a range of extensions to allow you to control and utilise its hardware to the full.

5) HARDWARE: interfacing, maintenance and other hardware-oriented details.

6) APPENDICES: Specification, character set, code conversion tables, etc.

The contents of each part are listed on its first page.

Each part is subdivided into chapters, numbered consecutively from the start of this manual. Within each chapter are a number of sections (section 1.1, section 1.2 etc) and subsections (subsection 1.1.1, subsection 1.1.2, etc).

1.3.4 Conventions

Within this manual, a few simple conventions are used to help clarify descriptions. These are mainly used to indicate text to type, as follows:

`DIR`	this is text which should be typed exactly as shown. If you prefer, you may type it as the equivalent small letters (d i r).
filename	this is a description of text to type, rather than the actual characters. In this case, you should type the name of a file (for example, PROGRAM1).
ENTER CAPS	this is a name or symbol representing the key that you should press or hold down.
[,A]	this is an optional part of the text ie text which may be omitted or typed, depending on what you want to do.
No File	text as it appears on the screen.

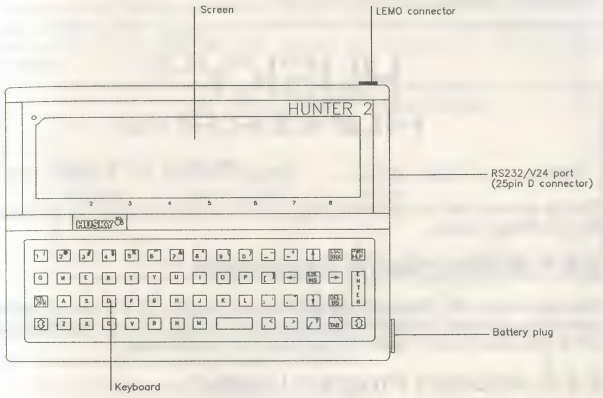
These conventions are often used in combination. For example:

`DIR` [*filename*] ENTER

This means type D I R and then a space, then either press ENTER or type the name of a file and press ENTER.

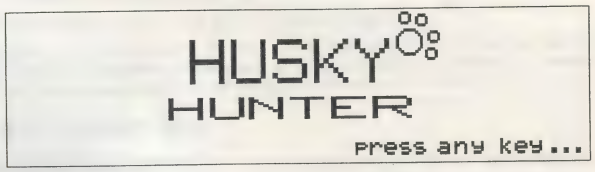
Chapter 2 Using Hunter 2

2.1 Introduction



2.2 Switching On

To switch on, press [PWR], which is at the top right hand corner of the keyboard. **Hunter 2** should respond with a 'beep' If not, check that the batteries have been installed or need to be replaced (see section 2.8). If all is well the following 'Welcome Message' should be displayed:



If all is not well and the above display does not appear, it could be for one of the following reasons.

2.2.1 Display Needs Adjustment

You can adjust the **Hunter 2** Liquid Crystal Display (LCD) to suit various operating positions. To do this, hold down the [CTL/FN] (control/function) key and press either ↑ or ↓ until you can read the display clearly.

2.2.2 Autostart Program Loaded

A program has been entered into **Hunter 2** to automatically run when ever it is switched on. If this is the case the operator should be prompted by **Hunter 2** or have been informed what to do next.

If neither of the above produce a response, check Chapter 3 for further information.

2.3 Selecting the File Manager

Having obtained the above 'Welcome Message', press any key to obtain **Hunter 2**'s File Manager. The File Manager is also called DEMOS 2.2 (Disk EMulation Operating System version 2.2).

The screen displays **Hunter 2**'s various system files, any of which may be selected by operation of the appropriate function key The exact appearance depends on the version of DEMOS and what options are installed, but this is a typical example:

```
          25 Oct  1989  12:50:02
** DEMOS 2.21 9G08h V4

BAS      .sys COMS     .sys CLCK     .sys
TERM     .sys EDIT     .sys
>

Dir  Stat Bas  Coms Clck Term Edit Keys
```

Only system files (.SYS extension) are displayed. Other files can be displayed by using the DIR command. The time display does not update to conserve power

The function keys are programmed as follows:

1. DIR (File directory)
2. STAT (File status)
3. BAS (BASIC Interpreter)
4. COMS (Initialise communications)
5. CLCK (Initialise clock)
6. TERM (Terminal emulation)
7 EDIT (Text Editor)
8. KEYS (Alternate function keys)

The 'KEYS' function switches the function keys to another set of functions, as follows:

1. DIR

2. STAT
3. INP
4. SAVE
5. TYPE
6. REN
7 SEND
8. KEYS

Functions can be selected by pressing the CTL/FN function key and the appropriate number key, or by typing the function name.

2.4 The System Files

Hunter 2's system files are not true files in the CP/M sense. They are built-in utility programs designed to support use of the computer The function of each is described briefly below For further information, refer to the index.

2.4.1 DIR - File Directory

Displays a list of filenames. The use of wildcards is supported. See 'DEMOS File Manager' in Part 2.

2.4.2 STAT - File Status

This function displays the file status, read/write or read only, file size and remaining file space. See 'DEMOS File Manager' in Part 2.

2.4.3 BAS - BASIC Interpreter

Access to **Hunter 2**'s BASIC Interpreter is provided by this option. See Part 3.

2.4.4 COMS - Initialise Communications

The parameters of **Hunter 2**'s industry standard RS232-compatible interface may be changed for compatibility with additional external equipment. See 'DEMOS File Manager' in Part 2.

2.4.5 CLCK - Initialise Clock

The date and time of **Hunter 2**'s internal calendar clock may be set with this function. The calendar and clock are maintained automatically once the correct date and time have been entered. See 'DEMOS File Manager' in Part 2.

2.4.6 TERM - Terminal Emulation

Hunter 2 emulates a simple CRT terminal. The communications parameters should be configured first to suit the host system. See 'Using Communications' in Part 2.

2.4.7 EDIT - Text Editor

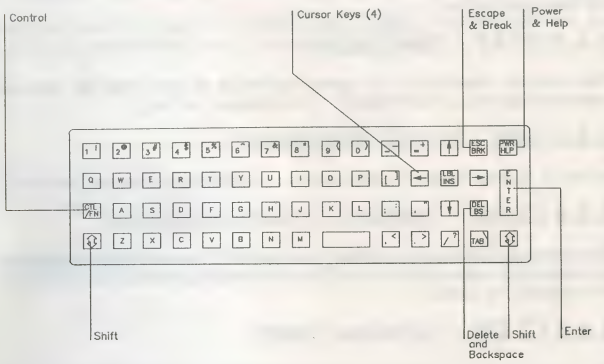
This activates the text editor See 'The Editor' in Part 2.

2.4.8 KEYS - Alternate Functions Keys

Operation of the 'KEYS' function displays the alternate function key display line, as follows:

DIR STAT INP SAVE TYPE REN SEND KEYS

Further operation of the 'KEYS' function will return **Hunter 2** to the original function key display line.



2.5 The Keyboard

Hunter 2's keyboard comprises 58 keys. Most are laid out in the standard QWERTY positions, but there are a few non-standard keys such as [PWR], [ESC], ⓘ, ⓙ, ⓚ and ⓛ

When held down, the keys will auto-repeat at about 8 characters per second.

2.5.1 Environment

Hunter 2's keyboard is built for extended use and long life. The keyboard will withstand assaults from water, coffee and mechanical shock, but it can be damaged by impact with sharp objects.

2.5.2 PWR and HLP

Used for switching **Hunter 2** on and off.

[PWR] is generally used to turn **Hunter** on and off again. If it has been programmed for HELP, press ⓢ [PWR] to turn off.

2.5.3 Turning on and off

Unlike most computers, **Hunter 2**'s ON/OFF switch (the [PWR] key) does not turn power off directly. This key is simply scanned by software, which makes the decision whether to turn off.

If a machine code program puts **Hunter 2** in an endless loop, it will be impossible to turn off the machine. To overcome this:

1) remove the main batteries
2) fit the main batteries again
3) hold down [CTL/FN] and ⓒ
4) turn on and release [CTL/FN] and ⓒ when **Hunter 2** beeps
5) type the emergency breakout code (by default, 56580)

2.5.4 ENTER

The [ENTER] key is used for terminating input from the keyboard. For certain inputs, such as replying to menus, [ENTER] is not used as **Hunter 2** will respond automatically to keyboard input.

[ENTER] is often called 'RETURN' in user programs. The functions are identical.

2.5.5 SHIFT

Hold down the ⇧ (SHIFT) to access the legends on the upper half of the keys. The keys A-Z produce upper case with the ⇧ key

2.5.6 Cursor Keys

The cursor control keys (↑, ↓, ← and →) perform a number of functions including adjustment of viewing angle, moving the virtual screen window as well as moving the cursor

2.5.7 DEL, BS and Backarrow

The [DEL], [BS] (backspace) and ← (backarrow) keys all perform the same function within DEMOS and BASIC ie delete the character to the left of the cursor. Some user programs in CP/M may use backspace differently by not deleting the character, but only moving the cursor back.

2.5.8 CTL

The [CTL/FN] key is similar to ⇧ in that it generates a new set of ASCII codes.

Hold down [CTL/FN] and press an alphabetic key to generate a control code. For example, to type Control A (which clears the screen), type [CTL/FN] [A]. To type Control G (which sounds a beep), type [CTL/FN] [G] (Note: this function is inhibited in the File Manager.)

Another use of [CTL/FN] is for adjusting the LCD display viewing angle. Press [CTL/FN] and either ↑ or ↓ to change the viewing angle to suit the user This function is always active.

2.5.9 FN and Function Keys

The [CTL/FN] key is also be used to generate key sequences known as function keys. These strings may be used for frequently used commands and are present within system programs (DEMOS, BASIC, etc.) They are operated in a similar fashion to control codes, but pressing a numeric key 1-8, giving the eight function keys.

The commands implemented are displayed on the bottom line of display (this may be turned off by using the [LBL] key)

Within BASIC programs the function keys may be initialised to perform new functions (see the KEY command, in Part 3).

2.5.10 ESC and BRK

[ESC] (Escape) is an upper-shifted key, used to break into and terminate execution of **Hunter** BASIC programs.

[BRK] (BREAK) is used to interrupt a BASIC program. See ON BREAK in Part 3. [BRK] is also available for CP/M programs and in terminal mode.

2.5.11 INS and LBL

[INS] will return the virtual screen window to its original position.

[CTL/FN] [INS] will reverse the operation of the ⇧ key on the characters A - Z, providing a CAPS LOCK function.

[LBL] switches the function line display on and off. [LBL] is an upper shifted key

2.6 The Screen

2.6.1 Introduction

Hunter 2 has a large LCD (Liquid Crystal Display) screen. It can present up to 320 characters on 8 lines of 40 characters each, just like a small video terminal.

The character set includes upper and lower case alphabets and special characters. Additionally, a graphics display of 240 x 64 pixels is supported.

Hunter 2's display is designed for use in bright sunlight where other displays (the red 'LED' or green 'vacuum fluorescent', for example) become invisible.

Because it works by contrast, rather than emitting its own light, it is perfectly visible no matter how bright the light is. But a word of caution: try to avoid leaving **Hunter 2** exposed to direct sunlight for prolonged periods - the delicate chemicals in the LCD can be damaged.

The 'cursor' shows where data entered on the keyboard will appear.

The screen is protected by an acrylic window, as strong as glass. Like glass, it can be scratched, so please take care. The window should only be cleaned with a chamois or lens cleaning cloth.

Hunter 2 is built to last - please take care of it.

2.6.2 Screen Modes

The screen can be used in two ways:

* Text mode
* Graphics mode

Text mode provides only character support with the standard 8 lines of 40 characters displayed. Internally, there is a "virtual screen", see section 2.6.4, supporting 24 lines of 80 characters. This mode can support any standard CP/M program, eg SuperCalc or dBase II. The operating system provides simple access to graphic routines from a CP/M program.

Graphics mode will support characters of five different sizes, reverse video and full graphic capabilities. There is no "virtual screen" associated with this mode.

2.6.3 Screen Control

The text screen will support the clear screen command and screen addressing. The screen is addressed throughout the virtual screen, that is the addressing range is from 0 to 79 horizontally and 0 to 23 vertically

Cursor addressing is a three character sequence sent to the screen handler It is of the form:

0FH, x coordinate, y coordinate

The x coordinate is a number in the range 0 - 79.

The y coordinate is a number in the range 0 - 23.

The screen may be cleared by sending a Control-A or 01H character to the screen handler

2.6.4 Virtual Screen

The eight by forty characters on the LCD are only one sixth of the number of characters contained within the virtual screen. The LCD acts as a window onto the virtual screen. The contents of the virtual screen may be inspected at will by using the cursor control keys in their shifted state.

Screen Size

Hunter 2's virtual screen is 80 characters by 24 lines. The virtual screen behaves very much like a terminal with automatic scrolling. When the cursor reaches the bottom line, the rest scrolls up losing the top line.

Moving the Window

The window may be moved over the virtual screen either manually by the user, or automatically by the virtual screen handler resident in the operating system and as such is transparent to the user

Manual Movement

1) Shifted cursor control keys (looking keys):

   By using the cursor control keys in their shifted state, the window can be made to move in the direction marked on the key, ie hold down ⇧ and press the desired cursor control key When this operation is performed it will be noticed that the cursor on the LCD (a blank flashing block) will change its position on the LCD and quite often disappear. This happens

because the cursor position refers to the position in the virtual screen, and when the window is moved the cursor position relative to the window position in the virtual screen will change.

2) Function "cursor control keys".

Press CTL/FN and ⊕ to see the right-hand 40 characters of the screen. Press CTL/FN and ⊖ to see the left-hand 40 characters again. This function is useful for easy reading of wide screen formats.

3) INS key.

Pressing INS will return the window to its original position prior to use of the looking keys, as outlined in (1) above.

Transparent Movement

The transparent movement of the virtual screen window is primarily controlled by inputs, both from the keyboard and the serial port. If **Hunter 2** is required to accept an input from, say, the keyboard then the window will automatically be moved to allow the viewing of the immediate area of the virtual screen centred around the present cursor position, thus displaying any prompts which may have been required by either a BASIC program or a CP/M type program.

It must be noted that printing text onto the virtual screen will not move the window However, if any part of the virtual screen which is presently in view on the LCD is changed by, say, a PRINT statement, then these changes will be reflected on the LCD, thus always giving a true representation of the virtual screen.

If **Hunter 2** is accepting input from the keyboard and the cursor reaches the forty-first position of the virtual screen, then the window will be moved laterally so that the cursor is now positioned mid-way across the LCD, as soon as the forty-first character is entered, thus displaying the previous 20 virtual screen characters and the following 20 virtual screen characters.

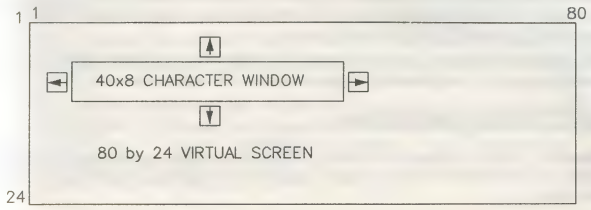
If **Hunter 2** is waiting for input and the looking keys are used to move the window, a valid input will cause the window to automatically revert to its original position prior to the use of the looking keys. This facility does not force the user to press INS in order to restore the original position of the window when in this mode.

For further information on cursor addressing, see LOCATE in Part 3.

NOTE: The virtual screen window does not operate in Graphics mode.

Moving the Cursor

To move the cursor in the required direction simply press the appropriate cursor control key.



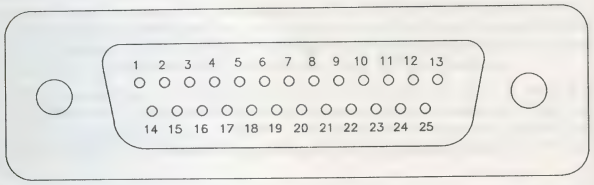
2.6.5 Screen Backlight

The **Hunter 2** can be fitted with an electroluminescent backlight facility, controlled from the keyboard, to provide screen illumination in low light conditions. Under 9G08, to turn the backlight on, press CTL/FN space, to turn it off, press CTL/FN space again.

On earlier **Hunters**, a similar backlight option was available, controlled by a switch mounted on the side of the case.

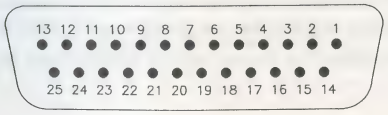
2.7 Communications

Hunter 2 has an industry-standard RS232-compatible serial communications port, and will talk directly to most computer peripherals, modems and other devices.



The port communicates via a standard male 25-way 'D' type connector The pin connections and other data are described in Part 2 (see 'Introduction to Communications').

NOTE: Never, never try to force a mating connector home. Check that the connector is the right way up (like this):



If the connector won't fit easily, it probably won't fit at all.

FIND OUT WHY

Hunter 2's communication format is usually set up by its applications program for specific situations, and does not require operator attention.

Sometimes, a user program will present a 'menu' selection for operator choice. In this event, selection of the desired communications mode will automatically set the communications parameters without further action.

Communications parameters can also be set by using a menu program, which is built in to **Hunter 2**. This can be accessed from the File Manager or by a machine code call from a user application program. See 'Introduction to Communications', in Part 2, for details.

2.8 Batteries

Hunter 2 is powered by batteries. It runs for a long time on a set of standard 'AA' size cells, which are widely available.

A second, rechargeable, battery is permanently built into **Hunter 2**. This is kept charged by the main batteries and is present to prevent data loss if the main batteries are removed. This battery should never need to be replaced.

2.8.1 Battery Installation

> SAFETY WARNING
> **ENSURE BATTERIES ARE CORRECTLY INSERTED FOR SAFE OPERATION. DO NOT USE A CHARGER WITH ALKALINE BATTERIES.**

The batteries are spring-loaded to make contact and retained by a threaded plug. Install the batteries as follows:

* ONLY USE A COIN in the battery plug. Screwdrivers, etc., will damage the slot and **Hunter 2**'s appearance.
* Feed in the cells, **positive inwards**. Don't drop the 'AA' cells in vertically - they can be damaged. Instead, slope **Hunter 2** slightly
* Take the battery plug and, with finger pressure only, press into the battery compartment and turn.
* Only when the thread is started, use a coin to screw the plug home.

The battery plug should stand about 2mm proud from the case wall.

Turn the plug clockwise to insert it, anti-clockwise to remove it.

2.8.2 Main Battery

Disposable Cells

We strongly recommend the use of Alkaline-Manganese batteries similar to Mallory MN1500.

We do not recommend the use of cheaper zinc-carbon cells for routine operation, since these can suffer electrolyte leakage, have shorter lives and can introduce technical problems.

Alkaline cells will produce approximately 45 hours of use when running a typical application program, and almost indefinite storage life. There is no need to remove cells from **Hunter 2** during storage or shipment but remember that capacity will reduce with time.

NOTE: DO NOT USE A CHARGER WITH ALKALINE BATTERIES.

Rechargeable Cells

Only Nickel-Cadmium (NiCad) cells are to be used. Under no circumstances must any other type of rechargeable cell be installed.

2.8.3 Low Battery Warning

Hunter 2 is designed to provide adequate warning of impending battery failure and will protect user's data, even in the worst situations.

When **Hunter 2**'s batteries become low, it will give warnings to the user. Since it is usually essential that all data and programs are secured, the user should switch off as soon as practicable and either replace the batteries with a fresh set, or if rechargeable batteries are fitted, put it on charge.

Low Battery warnings are repeated as follows:

* every 5 seconds, if **Hunter 2** is switched on, but quiescent.
* after every keyboard key depression.
* every 0.5 seconds, if characters are being received on the RS232 port.

If ten consecutive warnings are ignored, **Hunter 2** automatically switches off. When this occurs, **Hunter 2** is placed automatically in 'CONT' mode. This means that when power is restored, program execution will continue from precisely where it was prior to the interruption.

If an attempt is made to use **Hunter 2** without replacing or recharging the batteries, it will display the Battery Low warning and turn itself off again. It is worth commenting that flat batteries can partially recover after being switched off such that operation could continue for a short period. It is not recommended that this should be done in practice, particularly with rechargeable batteries which could then become totally exhausted. In the case of NiCad batteries, this can damage the cells permanently

NiCad batteries discharge very rapidly when they are low, so you may not get 10 warnings before the power is too low for **Hunter 2** to operate.

Hunter 2 has two screen modes, text and graphic (see section 2.6). The battery low warning is slightly different in the two modes.

Text Mode

When a low battery is detected, the top line of the screen displays:

```
* WARNING BATTERIES ARE LOW *
```

Hunter 2 bleeps twice and then the original text is restored until the next warning.

The text screen contents are preserved and not affected by the warnings. Following power restoration, the screen is restored to its pre-warning state.

Graphics Mode

When a low battery is detected, the top line of the screen displays:

```
* WARNING BATTERIES ARE LOW *
```

in reverse video and using the smallest character set (CHAR0). This message stays on the screen between battery warnings. **Hunter 2** also bleeps twice. The warning message destructively overwrites the top line of the screen, the contents of which are lost and must be restored by the user's program.

2.8.4 Power Save

An automatic power conservation system is incorporated into **Hunter 2**, enabling the power to be removed from circuits when they are not in use.

Usually, the computer will enter power save mode whenever it is waiting for keyboard or serial input. However, a few programs defeat this facility by not using standard CP/M console calls and have a correspondingly higher overall power consumption. Please consult Husky Computers for advice in specific cases.

2.8.5 Battery Charging

Hunter 2 is optionally available with rechargeable cells and a mains (line) powered charger This arrangement allows cells to be recharged in **Hunter 2**, and for alkaline cells to be quickly substituted if the user forgets to recharge!

But BE CAREFUL - NEVER connect the charger when alkaline batteries are installed. They will NOT re-charge. Instead, they may explode, jam in the battery tube or leak corrosive chemicals.

If you lend your **Hunter 2** to someone else or send it for repair, check that it still contains rechargeable batteries before connecting the charger

See 'Accessories', in Part 2, for further information about using the battery charger

2.8.6 Continuous Connection

Hunter 2 can be powered permanently from the charger by simply leaving the unit connected. In this mode, the rechargeable cells will be kept 'topped up' by the charger despite the continuous current drain. Of course, a fully discharged battery will take longer to recover than if **Hunter 2** is powered down.

2.8.7 Auto Timeout

To conserve battery life, **Hunter 2** will automatically switch off after a period of inactivity. It will also sound a warning bleep every 2.5 minutes to remind the user that it is turned on. This timeout does not operate in Terminal Emulation mode. The timeout value can be modified by using the POWER keyword in BASIC (see Part 3) and has a default value of 5 minutes.

Chapter 3 Troubleshooting

3.1 Introduction

Hunter 2 is designed, built and quality tested for robustness in every sense including robustness of program execution.

Hunter 2's physical construction protects it against mechanical disturbance, while the conventional computer's Achilles heel - external electrical interference - is virtually eliminated.

Because of these factors, **Hunter 2** is most unlikely ever to mis-execute a user's program or behave in a way that is not thoroughly predictable, given sufficient insight.

3.2 Causes of Crashes

However, there are some specific situations that can cause **Hunter 2** to mis-execute its internal programs, or "Crash".

The commonest causes are:

3.2.1 Illegal System Calls

System calls from BASIC are not 'trapped' (this would restrict user programming) and can cause mis-execution if not valid.

3.2.2 Invalid User Machine Code

User assembly-level programs or subroutines can easily cause mis-execution by containing, for instance, invalid jump instructions.

3.2.3 Physical Degradation

Ingress of water, corrosion of internal parts, damage to components through excessive shock or persistent high level vibration can reduce electronic tolerances.

3.2.4 Operation Outside Specified Temperature Range

Hunter 2 is specified for operation in the range 0 to 55 degrees Centigrade (although a wide temperature range version is available for operation in the range -20 to 55 degrees Centigrade). Operation above this range reduces tolerances, while operation below this range severely restricts battery capacity. Sub-zero temperatures also slow LCD screen response time substantially. **Hunter 2** is unlikely to mis-execute program simply because of low temperatures, however, in the range 0 to -20 degrees Centigrade.

3.3 Symptoms of Crashes

"Crashes" are identified by these symptoms:

3.3.1 Keyboard Lock-out

Hunter 2's keyboard is entirely "soft" to allow user re-definition of key functions, **including** the power key If a program crashes, power control may be lost and **Hunter 2** will not switch off.

3.3.2 Clock Wipe-out

Following a mis-execution episode, **Hunter 2**'s calendar clock registers may be corrupted.

3.3.3 Program Corruption

The most serious consequence of mis-execution is when the microprocessor runs 'wild', writing data randomly to all parts of memory and, occasionally, corrupting user programs. This failure is generally catastrophic, and is unlikely to go undetected. The most likely consequence is that any attempt to RUN the corrupted program will result in keyboard lock-out (see above). Attempts to LIST corrupted programs may also result in lock-out.

3.3.4 Communications Parameter Corruption

Occasionally, crashes may corrupt the RAM locations used to store the communications parameters, resulting in odd displays in the communications setup menus.

If this happens, hold down one of the cursor control keys until a recognisable option appear If the parameters are badly corrupted, you can restore the default communications parameters by entering BASIC, typing POKE 63583,0 ENTER, then powering off and on again.

3.4 Recovery

3.4.1 Keyboard Lock-out

If keyboard lock-out (failure to power down) occurs, power down by removing the battery cap and then fit it again.

Hunter 2 should power-up again normally. If desired, try running the user program, but be ready to clear any lock-out that occurs using the above methods. If lock-out persists, type NEW ENTER after restoring operation.

3.4.2 Program Corruption

If a mis-execution has occurred, any user program stored in RAM memory must be viewed with suspicion. The safest solution, assuming that the keyboard is still usable, is to reload the program.

3.4.3 System Lock-out

In very rare cases, **Hunter 2** may fail to restart correctly after a mis-execution episode. This is possible in a program that uses auto-start and contains an invalid system call.

Any program corruption resulting could leave the auto-start flag set, but cause a lock-out when power-up is attempted.

The solution is to use the breakout sequence immediately after power-up, in order to reset the auto-start flag.

3.4.4 File Corruption

Because files are not stored in **Hunter 2**'s workspace area, mis-execution episodes are much less likely to cause corruption of this part of memory. In any case, the checksum facility will indicate if damage has occurred. Storing a 'back up' file in a different memory page virtually guarantees data integrity against the worst possible disasters.

A checksum error can result in a **No file** message being displayed in response to an attempt to use the file, even if the directory entry is present. In this event, the best solution is to load the file again, if you can.

If you cannot the load the file again, you can make the file readable again as follows, but note that this only corrects the errors in the file checksums; it will not correct any errors in the files themselves:

From the File Manager, type TYPE *filename* ENTER, which should display the message **No file**.

Enter BASIC and execute the following program:

```
10 DEFSEG=1
20 FOR X=256 TO 2048 STEP 32
30 A=PEEK(X+13)
40 B=PEEK(X+14)
50 IF A<>B THEN PRINT "FAILURE": POKE X+14,A
60 NEXT: DEFSEG=0
70 END
```


HUSKY HUNTER 2 TM

Part 2
Facilities

Chapter 4 DEMOS File Manager

4.1 Introduction

DEMOS is **Hunter 2**'s operating system. It provides a CP/M-compatible environment for storing and running CP/M programs and a range of commands for manipulating files and the **Hunter 2**'s hardware. These commands are described in section 4.5. For more details of CP/M compatibility and programming, see Part 4.

The CP/M compatibility provided by DEMOS means that a large range of commercially available CP/M software will run on **Hunter 2** with no modification.

4.2 Selecting DEMOS

After switching **Hunter 2** on, a 'Welcome Message' will normally be displayed. Press any key to select the DEMOS 'File Manager'.

```
          25 Oct  1989  12:50:02
** DEMOS 2.21 9G08h V4

BAS      .sys COMS     .sys CLCK     .sys
TERM     .sys EDIT     .sys
>

Dir  Stat Bas  Coms Clck Term Edit Keys
```

The > prompt shows that DEMOS is waiting for you to type a command.

If this does not happen, **Hunter 2** may be running an application in CONT mode (see CONT in section 4.5) or there may be something wrong with the unit (see 'Troubleshooting', in Part 1).

The File Manager is equivalent in function to CP/M's command line. The bottom line shows the titles of the soft (or function) keys, which can be used as a short cut when typing commands.

The File Manager allows you to:

* Run standard CP/M programs such as word processors and spreadsheets
* Run the **Hunter** BASIC Interpreter and applications written in **Hunter** BASIC

* Use DEMOS commands, to manipulate files, use communications and configure other aspects of the hardware

4.3 Using DEMOS

4.3.1 Typing Commands

You use DEMOS by typing commands when the screen is displaying the '>' prompt, as described in the next three sections.

You can also use the following control key combinations, in addition to the function keys indicated by the label line at the base of the screen:

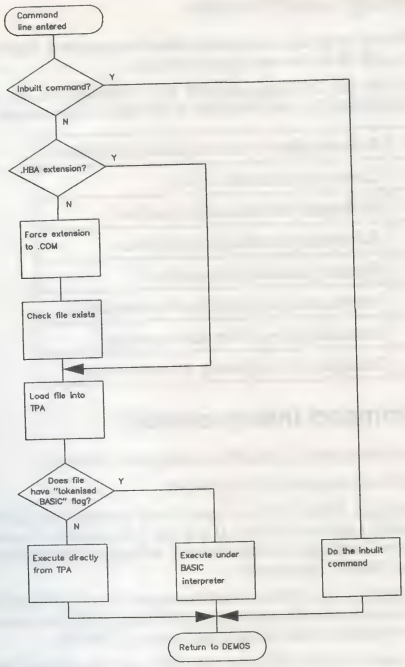
CTL/FN C	Reboots when at the beginning of the line.
CTL/FN E	Causes physical end of line.
CTL/FN H	Backspace one character position.
CTL/FN J	(Line feed) terminates input line.
CTL/FN M	(Return) terminates input line.
CTL/FN R	Retypes the current line after new line.
CTL/FN U	Removes current line after new line.
CTL/FN X	Backspace to beginning of current line.

4.3.2 Command Interpretation

Commands are interpreted by the File Manager as follows:

* Is it a built-in command? If it is, obey it!
* Is the extension .HBA? If it is, then it will assume that the file is **Hunter** BASIC, either tokenised or not. If it is not tokenised it will assume that it is ASCII and will attempt to tokenise it. It will then execute the program, via the **Hunter** BASIC Interpreter
* If the extension is not .HBA, it will interpret it as .COM, so that files with no extension entered will treated as .COM files. It will also prevent files with any other extension from being treated as a .COM file.
* DEMOS loads a .COM file into the TPA and checks the **Hunter** BASIC flag at the start of the file prior to execution. If this exists, then it will again assume **Hunter** BASIC and execute it as such.

Interpretation of commands by DEMOS



The only feature in any of the above which may lead to confusion is if you give a .COM file the extension .HBA and then attempt to run that. DEMOS will consider this an ASCII BASIC file, so BASIC will attempt to tokenise the file and then execute it, leading to unpredictable results, such as syntax error, system file error, etc.

4.3.3 Executing CP/M programs

To run a CP/M program, type its name. You may omit the .COM extension. The program loads into the TPA and executes without further intervention.

4.3.4 Executing BASIC programs

To run a **Hunter** BASIC program, type its name including the .HBA extension. **Hunter** BASIC is run automatically, loads the program into the TPA and executes it.

To enter the BASIC Interpreter without running a program, use the File Manager command BAS or CTL/FN 3. See Part 3 for further information on **Hunter** BASIC.

4.3.5 Continuous Execution

Introduction

The command CONT allows you to execute a program so that it will continue execution if **Hunter 2** is switched off and then switched on again. Power off may have been caused by execution of the POWER OFF function in BASIC, power-off timeout, or operation of the PWR key.

Command Syntax

The syntax of the CONT command is as follows:

CONT *filename* ENTER

where *filename* is the name of a BASIC or CP/M program file. The three character file type extension does not have to be entered if it is a .COM file, but must be included for BASIC files. For example, if a BASIC program file were called PROG1.HBA, the following command would load and run the program.

CONT PROG1.HBA ENTER

To run a CP/M program called PROGA.COM.

CONT PROGA [ENTER]

If *filename* does not have a .COM or .HBA extension, CONT will assume that is .COM, so if you specify a filename of PROG2.TXT, CONT will attempt to execute PROG2.COM. If this file does not exist a 'File not found' error will occur

Leaving CONT Mode

To escape from CONT hold down [CTL/FN][C], then power up. **Hunter 2** will beep twice and then wait for you to type the breakout escape code (which defaults to 56580). If you type this correctly **Hunter 2** will power down immediately and the next time you turn on, you will be taken to the File Manager via the Welcome screen, as normal.

If you enter the wrong code, **Hunter 2** will beep and continue execution of the program.

If you run a BASIC program by using CONT, you will also need to type the escape code to break out of the program by using the [ESC] key.

4.3.6 DEMOS Commands

The DEMOS commands are listed in section 4.5.

They are typed as the command name, followed (in some cases) by one or more arguments, usually filenames. Commands may be in upper or lower case alphabetic characters. Filenames may include wildcard characters, to specify groups of files.

You may also type the more common DEMOS commands by using the function keys (ie by holding down [CTL/FN] and pressing one of the digits 1-8), which types the command indicated by the label line at the bottom of the screen.

4.4 Filing

4.4.1 Introduction

DEMOS uses a large part of its RAM to implement a CP/M-compatible RAM disk. This can be used rather like a conventional disk on a desktop CP/M computer, but it is much faster

DEMOS is totally independent of **Hunter** BASIC (although you can, of course, use the RAM disk from BASIC, to store or access programs or data files).

4.4.2 Filenames

Files are named following the general format of CP/M. Names may be up to eight alphanumeric characters long, with an optional extension of a period and up to three more alphanumeric characters. Certain extensions have a special significance to the operating system, as follows:

.HBA	**Hunter** BASIC program
.COM	A CP/M compatible object code file which can be loaded and run independently of **Hunter** BASIC.
.DTA	A data file produced by a **Hunter** BASIC program.
TXT	An ASCII text file, produced, for example, by the DEMOS editor
.SYS	A 'pseudofile', used to provide **Hunter 2** facilities. These are generally stored in ROM.

4.4.3 Wildcards

DEMOS allows you to use wildcard characters ('?' and '*') in filenames.

The '?' character specifies a single 'wild' character. For example, suppose there are five files, with the names FILE1.TXT - FILE5.TXT. The command 'DIR' will list all the files on the disk. To list just these five files, you could use the following DIR command:

DIR FILE?.TXT [ENTER]

Similarly·

DIR A?FILE.COM [ENTER]

would list files with the names AAFILE.COM, ABFILE.COM, ACFILE.COM, etc.

The '*' character specifies any number of 'wild' characters in a filename or filename extension. For example, the following would list all files with the extension .HBA.

```
DIR *.HBA [ENTER]
```

Similarly:

```
DIR MYFILE.* [ENTER]
```

would list files called MYFILE.TXT, MYFILE.COM, MYFILE.HBA or even just MYFILE (no extension).

You may also use a combination of wildcards, like this:

```
DIR ??FILE.* [ENTER]
```

would list files with names like MYFILE.TXT, MYFILE.COM, MYFILE.HBA, AAFILE.COM, ACFILE.COM, A4FILE.COM, MRFILE.HBA etc

Wildcards may be used with File Manager commands other than DIR. For example, with ERA.

```
ERA BIG*.* [ENTER]
```

this will erase all files with names that start with 'BIG' and any extension for example, BIGGER.FIL, BIG.HBA or BIGGLES.COM would all be deleted.

(In this case, **Hunter 2** will query the erasure of each file in turn, as a precaution.)

4.4.4 Data Integrity

Hunter 2 uses a sophisticated power management system to preserve the information stored in this disk while it is turned off.

To minimise the risk of accessing data which has become corrupted, each time a file is modified DEMOS calculates a checksum for the file and stores it in the directory The next time you attempt to access ('open') the file, DEMOS will calculate a checksum for it again and only allow you to use the file if the two checksums are the same. (If not, a "No file" message will be displayed.)

4.5 The DEMOS Commands

4.5.1 BAS

Form

BAS [ENTER]
or
[CTL/FN] 3

Explanation

Execute **Hunter 2**'s BASIC Interpreter

Examples

When the BASIC Interpreter has been entered the following screen is displayed:

```
Hunter Basic Interpreter
Ready



File Run  Edit List Save Load Kill Syst
```

Remarks

Hunter 2's BASIC Interpreter is executed from ROM. If the last program was a CP/M program then BASIC performs an automatic NEW, clearing its variables and workspace. If the last program was a BASIC program, this will be in the workspace and any variables that were defined will remain defined. For complete information on BASIC programming refer to Part 3.

4.5.2 CLCK

Form

CLCK [ENTER]
or
[CTL/FN] 5

Explanation

Set the internal calendar clock to the desired date and time.

Examples

CLCK enters the calendar clock initialisation screen:

```
        25 Oct   1989  12:50:02
      Time Setting Mode
```

Remarks

To set the internal calendar clock proceed as follows:

a) Using the ⊕ and ⊕ keys, move the cursor to the item you wish to alter. (You cannot alter the seconds in this way· they are set to zero when you leave this screen.)

b) Using the ⊕ and ⊕ keys, step the value selected until it is correct.

The months step in the following manner:

Nov, Dec, Jan etc. , Cursor up
Jan, Dec, Nov etc. ; Cursor down

c) Set the 'minutes' to one minute ahead of the present time.

d) Press [ENTER] and the following will be displayed:

```
Press ENTER to synchronise
```


e) When the time code, pip tone, or other reference arrives press [ENTER] to reset **Hunter 2**'s clock to zero seconds, in exact synchronisation with the external reference.

The calendar automatically accounts for leap years.

If you attempt to specify an incorrect day value, eg 31st April, **Hunter 2** will display the warning:

```
Error - Incorrect day value
```

Correct the error before synchronising the clock.

Hunter 2's clock can be used to time-stamp entries, automatically label printouts or even initiate program execution automatically like an alarm clock.

See Part 3 for details of the TIME$, DAY$ and DATE$ functions and Part 4 for details of CP/M clock calls.

NOTE: In versions of the operating system earlier then 9G08, there was an error which meant that the year might not increment automatically at year end. This error has been corrected.

If you are writing application programs, remember to include facilities to allow users to adjust the clock when required eg when changing time zones, or when adjusting for British Summer Time, in the UK.

4.5.3 COMS

Form

COMS [ENTER]
or
[CTL/FN] 4

Explanation

Initialise the communications parameters.

Examples

Two screens are used to select the communications parameters as follows:

```
        Transmission Parameters
  Rate-1800  Prtcl-none      Pty-none
CTS-n DTR-n LF-n Echo-y T/O-no Null- 0
      press ENTER if acceptable.
```

```
         Receiving Parameters
  Rate-1200  Prtcl-none      Pty-none
RTS-off  DSR-n DCD-n  T/O-no Serig-off
      press ENTER if acceptable
```

Remarks

See Chapter 6 for information on selecting communication parameters.

NOTE: Communication parameters are not reset by powering off and on.

In CP/M, the serial communications interface is assigned the following logical devices:

Transmit - LST:, PUN·
Receive - RDR.

4.5.4 CONT

Form

CONT *filename* [ENTER]

Explanation

Execute program with 'continue' option.

Examples

CONT PROG1 [ENTER]

This will load and run the CP/M program 'PROG1.COM'

The three character file type extension does not have to be specified for .COM files but must be included for **Hunter** BASIC files, for example, a **Hunter** BASIC program has the name 'SORT.HBA', the following would load and execute the program via the BASIC Interpreter·

CONT SORT.HBA [ENTER]

CONT should not be used with BASIC programs that include commands such as ON POWER OFF In general, CONT should be reserved for use with COM files.

Remarks

If **Hunter 2** is switched off or power down occurs for any other reason while the program is executing, when it is switched on again, the program will continue execution from the point at which power was removed.

To escape from CONT mode when power is off, hold down [CTL/FN] and C, press [PWR] to power on, and wait until you hear two 'beeps' before releasing the keys. Now type the 'escape code', a special security number, the default for which is 56580. If you type the code incorrectly, the **Hunter 2** will 'beep' and the program will continue executing.

In CONT mode, operation of [CTL/FN] C and [BRK] is also affected: you must type the escape code to stop the program executing.

4.5.5 DIR

Form

DIR [*filename*] [ENTER]

Explanation

Display the file directory

Examples

DIR [ENTER]

```
ONEPROG  .HBA TWOPROG .COM FINAL   .HEX
ANOTHER  .1
```

DIR can also be used with wildcards as follows:

DIR FILE?.TXT [ENTER]

```
FILE1    .TXT FILE2    .TXT FILE3   .TXT
FILE4    .TXT FILE5    .TXT
```

or

DIR A?FILE.COM [ENTER]

```
AAFILE   .COM ACFILE   .COM A4FILE .COM
```

or

DIR *.HBA [ENTER]

```
PROG1    .HBA PROG2    .HBA START   .HBA
```

Remarks

A combination of wildcard characters can also be used.

If the directory is too long to fit on the display screen, use [⇧][↑] to scroll back the virtual screen to see the earlier entries.

The virtual screen contains 24 lines, if a file directory extends to more than 24 lines, the top lines will scroll off the screen. Use of wildcards may help in limiting the number of directory lines displayed.

4.5.6 EDIT

Form

`EDIT` [*filename*] ENTER

Explanation

Edit an ASCII text or Husky BASIC (.HBA) file.

Examples

`EDIT PROG1.HBA` ENTER

This will edit a Husky BASIC file.

`EDIT TEXT` ENTER

This command will edit an ASCII file TEXT TXT The .TXT extension is assumed if none is specified.

`EDIT NEWS.DAY` ENTER

Edit the ASCII file of the above name.

Remarks

For further information on the editor, see Chapter 5.

NOTE: Selecting the editor will remove the current BASIC program ie the next time BASIC is selected, the workspace will be blank.

4.5.7 ERA

Form

`ERA` *filename* ENTER

Explanation

Erase the specified file(s).

Examples

`ERA MYFILE.HBA` ENTER

ERA may also be used with wildcards as follows:

`ERA MY*.*` ENTER

will erase any files starting with 'MY' and any extension.

Remarks

Each specified filename will be displayed in turn for the user to enter Y to delete the file or N to retain the file.

NOTE: You cannot easily recover a file once you have deleted it. (See Appendix VI for a utility program which can be used to recover deleted files under certain circumstances.)

4.5.8 EXECUTE

Form

EXECUTE (ENTER)

Explanation

Execute the current program in the TPA (RAM page 0), entering it at 0100H.

Examples

EXECUTE (ENTER)

Remarks

This command will normally only be used after loading a program by using the LOAD or INP command.

However, you may also use it to execute a program again once it has been loaded, provided that it does not alter itself in any way.

4.5.9 FORMAT

Form

FORMAT (ENTER)

Explanation

Re-sets **Hunter 2**'s filing system.

Examples

FORMAT (ENTER)

As formatting the RAM disk is such as serious step, the File Manager first checks that this is what you intend:

```
FORMAT?: Are you sure (Y/N)
```

Remarks

When FORMAT is executed all files are deleted from **Hunter 2**'s memory, leaving the user with only the built-in system files. This command should be used with caution. It is generally only required if a machine code program loaded into **Hunter 2** has crashed causing destruction of files and directories.

If (Y) is typed in response to the above question the File Manager will re-set **Hunter 2**'s filing system. Any files in the storage area will be lost. Also any programs loaded into the BASIC workspace are deleted.

If any other character is typed **Hunter 2** will return to the File Manager, without erasing any files.

FORMAT also resets the emergency escape code to 56580.

4.5.10 INP

Form

`INP` [*n filename*] ENTER

Explanation

Input a file via the serial port.

INP can be used to load files in either Intel Hex format, with checksums etc., or as 8-bit data but without any error checking. The second method of loading is faster than using Intel Hex.

Loading Intel Hex files

Files to be loaded must be in Intel Hex format (see Appendix VII). If they are command files, overlay files or data files, you can convert them to this format by using the LOAD utility reproduced in Appendix VI.

To load Intel Hex format, the command is:

`INP` ENTER

Hunter 2 converts the received data back to the original binary form and then puts the data received in the TPA (RAM page 0). When loading has finished, the number of 256 byte blocks loaded is displayed. You may now save the program as a file using the SAVE command, or execute it (if it is a COM file) using the EXECUTE command.

Note that **Hunter 2** only counts the number of blocks, not the memory required by the file. If there is a gap in the data, the number of blocks loaded will not represent the memory occupied by the file and you should take care when saving the TPA.

Loading 8-bit format

Data is loaded directly into a specified file in blocks of 256 bytes. The command format is:

`INP` *n filename* ENTER

where *n* specifies the maximum number of 256 byte blocks to load. For example, to load a 14k overlay file and save it as OVERLAY1.OVR.

```
INP 56 OVERLAY1.OVR ENTER
```


The communications must have parity set to 8-bit and SERIG (Serial ignore character) set to off.

The number of blocks must be specified because there is no end of file character If you specify too many blocks, the screen will display 'waiting' after data reception ceases and you can press ESC to terminate reception and save the file.

If you are not sure of the number of blocks, simply specify 999, wait for the **WAITING** message, and press ESC.

Remarks

When **Hunter 2** is loading data the following messages will appear on the screen:

```
WAITING
LOADING
WAITING
LOADING....
```

Hunter 2 responds to each pause in transmission with a **WAITING** message to indicate that data is not being received.

There are 4 blocks to 1K byte.

Because INP clears the receive buffer when it is invoked (to remove spurious data), it is important to execute INP on the **Hunter 2** before the transmitting device begins to send data.

4.5.11 KEYS

Form

KEYS [ENTER]
or
[CTL/FN] 8

Explanation

KEYS switches to the other set of function key definitions, as indicated by the label line display Selecting this command again switches back to the first set.

Examples

Set 1 is:

```
DIR STAT BAS  COMS CLCK TERM EDIT KEYS
```

Set 2 is:

```
DIR STAT INP  SAVE TYPE REN  SEND KEYS
```

Remarks

DIR and STAT are the most commonly used keys and appear in both sets, at convenient locations.

DIR	Directory
STAT	File status
BAS	BASIC Interpreter
COMS	Communications parameters
CLCK	Set clock
TERM	Terminal emulation
EDIT	Text Editor
INP	Load a file
SAVE	Create a file from memory
TYPE	Display an ASCII file
REN	Rename a file
SEND	Transmit files

4.5.12 LOAD

Form

LOAD *filename* [ENTER]

Explanation

The file is loaded into the TPA (RAM page 0) ready for execution (using EXECUTE) or dumping through the serial port in Intel Hex format (using SEND).

Examples

LOAD PROG1.COM [ENTER]

The program PROG1.COM is loaded into the execution memory

Remarks

Files are loaded into location 0100H upwards. Normally, this facility will be used to dump the file in Intel Hex format.

4.5.13 REN

Form

REN *filename1=filename2* ENTER

Explanation

Changes the name of *filename2* to *filename1*

Examples

REN NEW.NAM=OLD.NAM ENTER

This will rename the file OLD.NAM to NEW.NAM.

Remarks

The filenames must not include wildcard characters.

4.5.14 SAVE

Form

SAVE *n filename* ENTER

Explanation

Create a file from current memory

EXAMPLE

SAVE 128 BIGFILE.COM ENTER

This will create a file 32k bytes (128 blocks) in length.

Remarks

A program currently residing from 100H in the TPA (RAM page 0) is transferred to the specified file. The number of 256 byte blocks is specified by *n*.

This will normally be used to save a file that has been input from the communications port, in Intel Hex format. (See INP)

The maximum program size is 54K, so the maximum value for *n* is 215.

4.5.15 SEND

Form

SEND *filename* [A] ENTER
or
SEND *n* ENTER

Explanation

Outputs data via the serial port.

Examples

To output a file in 8-bit format:

SEND DEMO.COM ENTER

To output an ASCII file to, say, a printer·

SEND LETTER.TXT A ENTER

To output in Intel Hex format, use LOAD to load the file into the TPA (RAM page 0), then type:

SEND *n* ENTER

where *n* is the number of 256 byte blocks required to be sent.

Remarks

To transmit in 8-bit format, remember to set Parity to 8-bit.

Transmission can be aborted with the ESC key.

ASCII output is the same as 8-bit output except that output ceases when a Control-Z character (1AH) is read from the file, to avoid printing data in the last block that is beyond the end of the file. (Control-Z is used as an end-of-file marker in text files.)

If using 8-bit output for a .COM or a **Hunter 2** .HBA file, you must turn off the LF and NULL options in the communications menu, to prevent insertion of spurious LF and NULL characters after CR characters, which would corrupt the output.

4.5.16 STAT

Form

STAT [*filename*] ENTER
or
CTL/FN 2 [*filename*] ENTER

Explanation

Extended file directory display.

Examples

STAT ENTER

will display the space remaining on the RAM disk.

STAT FIRSTFIL.COM ENTER

will display data about the specified file, like this:

```
Recs   Bytes   Ext   Acc
  64     16K     1   R/W   :FIRSTFIL.COM
24K bytes available
```

Where the number in the 'Recs' column is the size of the file in 128 byte records, the number in the 'Bytes' column is the size of the file in Kbytes, the number in the 'Ext' column is the number of 16K extension blocks and the data in the 'Acc' column is the access allowed to the file (R/W - Read Write, R/O Read Only).

Remarks

STAT may also be used with wildcards. For example:

STAT *.HBA ENTER

Will display the status of all **Hunter** BASIC files stored in memory.

4.5.17 TERM

Form

TERM [ENTER]
or
[CTL/FN] 6

Explanation

Hunter 2 emulates a CRT terminal.

Examples

TERM [ENTER]

Remarks

In terminal emulation mode, **Hunter 2** can be used as:

* A remote dialup terminal.
* A peripheral to other computer systems.
* A portable 'Telex'

See section 7.3 for further information.

Two function keys are implemented in TERM:

[CTL/FN] 4 - COMMS set up
[CTL/FN] 8 - Exit to system

The type of terminal to be emulated is determined by the setting of the communication parameters. See Chapter 6 for details.

Automatic power down does not operate in Terminal Emulation. This prevents inadvertent dropping of communications lines.

TERM may be used from user programs (see section 7.3).

4.5.18 TYPE

Form

TYPE *filename* [ENTER]

Explanation

Display the contents of an ASCII file.

Examples

TYPE FILE.TXT [ENTER]

This will output the filename FILE.TXT to the screen.

Remarks

Scrolling may be halted by using [CTL/FN] S and restored with any other key. Any character typed while the screen is scrolling, other than [CTL/FN] S will abort the display.

Typing non-ASCII files may give confusing results.

4.5.19 /DO

Form

/DO [*filename*] ENTER

Explanation

Execute a **Hunter** BASIC program from the RAM disk.

Examples

/DO FILE.HBA ENTER

where FILE.HBA is a **Hunter** BASIC program.

Remarks

This command, which is unique to the **Hunter 2**, will run a **Hunter** BASIC program directly from the filespace RAM. This leaves the whole of the TPA for storing data or machine code in arrays.

This feature is not intended for development of programs, but is to enable the final version to have as large a data area as possible.

/DO will only execute the program correctly if it is stored contiguously in the first page of disk memory This limits the maximum program size to 46K. To ensure that the program is stored correctly, observe the following procedure:

* Use FORMAT to erase all files.
* Load the program from the bulk storage media (disk or host system) by using the BASIC LLOAD command.
* Save the program using the BASIC SAVE command.

This procedure stores the program contiguously starting from the bottom of the first file page of RAM. You may now load any other files that you want on the **Hunter 2**'s RAM disk.

You may now execute the program using /DO.

To make the program execute continuously, use ON POWER RESUME within the BASIC program (see ON POWER RESUME, in Part 3).

Do not attempt to edit a program running in this way as the editor expects to find the program in the TPA.

4.6 DEMOS Error Messages

The following are operating system error messages which can occur in DEMOS:

Aborted

You have pressed the ESC key while executing a TYPE or INP command.

Bad Name

Whenever an invalid filename format is specified.

Disk Error

Due to a file having been corrupted, eg an incorrect checksum for the file.

Disk Full

At any time when trying to write to a file when there is no space remaining in the file space.

Memory Overflow

Due to:

* The text equivalent of a BASIC tokenised file exceeding the available space in RAM page 0, when using EDIT
* Excess number of blocks in SAVE.
* Excess number of blocks in SEND.

No File

Due to:

* Filename not specified, in an EDIT command
* The specified file does not exist, in an ERA command
* The old filename does not exist, in a REN command
* The specified BASIC file does not exist, when trying to load and run the file from DEMOS
* No files in the **Hunter 2**, when you enter STAT *.*
* There is no .COM with the name entered
* The specified file has become corrupted

Syntax

Due to:

* Missing '=' between the filenames in REN
* Missing number of blocks in SAVE
* Missing space between number of blocks and filename in SAVE
* Missing number of blocks in INP
* Missing space between number of blocks and filename in INP
* Missing number of blocks in SEND
* Missing ENTER immediately after the filename in SEND

Chapter 5 Editor

5.1 Introduction

Hunter 2's built-in text editor provides practical text handling facilities.

It may be used for the creation and alteration of standard ASCII text files for reports, correspondence, etc. This feature is invaluable for text entry 'on the move' in trains, planes or any other form of transport. Text may then be easily printed out on return to the office.

Hunter BASIC files, which are encoded to save memory and execution time, may also be edited directly. These files are recognised by their use of the .HBA suffix. It is important, therefore, to maintain the .HBA suffix convention when attempting to edit BASIC programs.

Encoded source files for other interpreted programs such as MBASIC cannot be edited with the Editor.

Selecting the Editor will remove the current BASIC program ie the next time BASIC is selected, the workspace will be blank.

5.2 Selecting the Editor

The editor can be invoked from the File Manager or directly from BASIC. In either case, procedures are similar but, on exit, the user will be returned either to the File Manager or BASIC, as appropriate.

5.2.1 Entry from the File Manager

From the DEMOS File Manager, type:

`EDIT` *filename* ENTER
or
CTL/FN 7 *filename* ENTER

For example:

`EDIT myfile.txt` ENTER

If the specified file already exists, it will be loaded into the editor If the file does not exist, it will be created on exit from the editor provided text entry has been made.

It is essential to enter a filename. If you omit the extension, TXT will be assumed and the file will be assumed to be ASCII text.

Assuming myfile.txt does not exist, the screen will show:

```
COL:00 INS:on Caps:off



Caps Find Cins                Save Ldel Exit
```

On the top line there are three items of information:

COL:00 This shows the column which the cursor is in. It is in the range 0 to 79. As the cursor moves, it is kept updated and may be used for aligning tables, indents in text, etc.

Column 0 is the left of the virtual screen. Column 79 is the extreme right of the virtual screen.

INS:on This shows whether character insert is on or off.

If insert is on, any text entered will be placed at the cursor position and any text already present will be moved.

With insert off, the display will show:
`INS:off`
In this case, entries will overwrite existing text.

When you are creating a new file or entering text at the end of an existing file, insert defaults to on. At other times, it defaults to off.

Caps:off This shows the state of the CAPS lock function.

In CAPS LOCK mode, alphabetic keys type capitals unless used with ⇧, when they type lowercase letters. This is NOT a 'Shift Lock' function.

You may now enter and edit text as required. The cursor keys provide up, down, forward and backspace functions. Lines are terminated by ENTER which is equivalent to the CR (Carriage Return) character (ASCII 13).

If the file already exists, the first few lines of text will appear on the screen, like this:

```
COL:00 INS:off Caps:off
 This is a demonstration of entering tex
the text editor. On this screen it can b
we only display the top left of the file

 Paragraphs are generally indented, this
no problem to the Hunter text Editor.
Caps Find Cins                Save Ldel Exit
```

This is the beginning of the file, showing the left-hand forty columns. Text editing may now proceed as required.

NOTE: The editor works on a copy of the file, held in the TPA. You must save the text explicitly to change any existing file.

Filenames

If no file extension is supplied, the editor will default to the file extension TXT and assume an ASCII text file. If the extension is .HBA, the editor will convert the tokenised file to ASCII on entry and re-tokenise it on exit.

If a tokenised BASIC file is specified using a different extension, the editor will still recognise it as tokenised BASIC and convert it similarly

5.2.2 Entry from BASIC

When in BASIC (see Part 3), you may use BASIC's EDIT command to edit the current program or a specified file.

To edit the current program, type:

`EDIT` ENTER
or
CTL/FN 3 ENTER

There will be a slight delay while the BASIC program is converted to ASCII. The screen will then appear as shown above and editing can proceed as required.

Alternatively, to edit a different file, type:

`EDIT` *filename* ENTER
or
CTL/FN 3 *filename* ENTER

Where *filename* is a string constant or variable. (See Part 3 for more details.)

If the file extension is omitted, the .HBA extension will be assumed.

When exiting from the Editor after entering from BASIC, use of EXIT will cause the ASCII character representation used by the Editor to be tokenised ready for execution by BASIC. This procedure can take a few seconds for larger programs.

Any unnumbered lines will not be tokenised and consequently will be lost. Any duplicated line numbers will cause the earlier entry to be lost.

5.2.3 Automatic Cont Edit

When you select the editor, the **Hunter 2** is forced into CONT mode. This applies even if the Editor has been initiated but is still converting a BASIC tokenised program into its text equivalent and the normal Editor screen is not yet displayed.

If **Hunter 2** is now powered down, for whatever reason, then on power up it will re-enter the Editor and continue operation as if it had never been turned off. This applies in all cases eg midway through a search, writing to a file or exiting to BASIC and recreating a BASIC tokenised file from its text equivalent.

This feature ensures that all data entered into the **Hunter 2** whilst in the Editor is always accessible to the user. On exit from the Editor, either by 'Exit' or 'Save', the **Hunter 2** is forced out of CONT mode automatically. Thus, if the **Hunter 2** is powered down after this, it will power up normally.

5.2.4 File Size Limit

The maximum size file that the editor can handle is 54K bytes. If you attempt to edit a file larger than this, a warning message will be displayed and the operation will be aborted.

The file will not have been changed in any way and the **Hunter 2** will not be in CONT mode.

If attempting to edit a BASIC program, the program is converted from its tokenised form to less compact ASCII text for editing. This means that the maximum size program file that you can edit is approximately 36K bytes.

These size limits are unlikely to cause any problems in practice.

5.3 Editor Operation

5.3.1 Introduction

The Editor is incorporated in the operating system as a built-in function and operates as a file-based system.

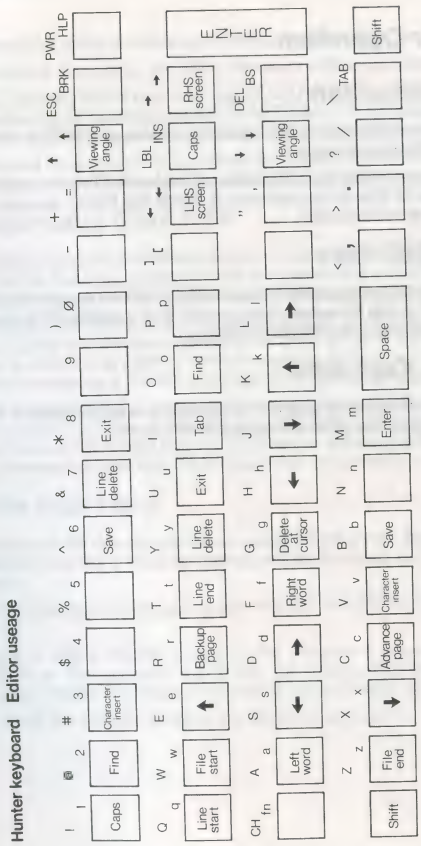
When a file is loaded into the Editor, it is placed in the TPA, the 54K workspace used by **Hunter 2**. This is the only area in which the Editor operates on a file. The whole file must be loaded.

5.3.2 BASIC Files

If you edit a BASIC program it is converted to ASCII on entry to the editor and converted back to the tokenised form on exit. This causes a slight delay, which will be more noticeable for longer programs.

5.3.3 The Keyboard

The function performed by each key when used as a control key (ie hold down CTL/FN and press the appropriate key) is as follows:



All the function keys are replicated as control keys which conform broadly to the layout used by WordStar

The Editor uses the virtual screen, so the cursor keys may be used to scan around the text in the virtual screen.

The label line cannot be turned off with LBL when in the Editor

If you hold down a key, it will 'autorepeat' after a short delay If you do this, the screen may lag behind your actions. To avoid deleting text by mistake, you should not hold down any of the character delete keys.

5.3.4 Line Length

The Editor supports lines of greater than 80 characters by wrapping round onto the next line.

5.4 Editor Commands

5.4.1 Introduction

Editor commands are typed by using the CTL/FN key in conjunction with the letter keys and number keys. These may in general be typed at any time.

Each group of commands will be described separately. Refer to section 5.3.3 for details of the keys used for each command.

Control and function keys are denoted by CTL/FN *character*. For example, Control G is shown as CTL/FN G and function key 2 as CTL/FN 2.

5.4.2 ARROWS

Form

CTL/FN E; CTL/FN K, ↑ - Up arrow
CTL/FN X; CTL/FN J; ↓ - Down arrow
CTL/FN S; CTL/FN H, ← - Left arrow
CTL/FN D; CTL/FN L; → - Right arrow

Use

These keys move the cursor non-destructively in the required direction. They may be used to place the cursor over any character position which is in use.

These keys will not place the cursor outside the text area, ie before the beginning of the file, after the end of the file, or past the end of a line.

Notes

The left arrow key will scroll to the previous line if the beginning of the line is reached. Similarly, the right arrow key will scroll to the next line if the end of the line is reached.

The cursor keys (↑, ↓, ← and →) have a number of other functions:

- When 'shifted', these keys scroll the screen window across the virtual screen.
- When pressed with CTL/FN, they have the following functions:
  - ↑ and ↓ adjust display contrast.
  - ← and → display left and right halves of the virtual screen.

5.4.3 CAPS LOCK & TAB

Form

CTL/FN LBL - Caps Lock
CTL/FN 1 - Caps Lock
TAB - Tab
CTL/FN I - Tab

Use

CAPS LOCK causes all alphabetic keys to type capitals unless used with SHIFT. This function is shown on the top line of the display. Pressing CAPS LOCK a second time turns the function off.

TAB moves the cursor onto the next tab position on the current line. These positions are set every 8 characters ie at column 8, 16, 24, etc.

Notes

CAPS LOCK is useful for entering programs.

5.4.4 CHARACTER INSERT

Form

CTL/FN 3 - Character Insert
CTL/FN V - Character Insert

Use

CHARACTER INSERT enables and disables character insert mode. The mode is shown on the top line, like this:

```
INS:on
```

When turned off, characters entered overtype existing text.

When turned on, new text is inserted before the cursor position. This is useful for adding characters to the middle of a line.

Notes

An exception to the above rule is ENTER (carriage return). With insert on, a carriage return is added as with any other character, causing a break of line if the cursor is in the middle of the text. With insert off, the cursor is moved non-destructively to the beginning of the next line unless the cursor is at the end of the file, when it is added to the file.

5.4.5 DELETE

Form

DEL - Delete
BS - Back space
CTL/FN G - Delete ahead

Use

Delete destructively moves the cursor back over the previous character. The character to the left of the cursor is removed and the rest of the line is moved one character to the left.

Backspace operates exactly as back arrow It moves the cursor non-destructively one character to the left.

Delete ahead destroys the character on which the cursor is positioned. The cursor remains stationary and text ahead of the cursor is moved towards it.

Notes

Deleting a carriage return joins the current line to the next line.

WARNING: Because **Hunter 2** has a 32 character keyboard input buffer, 'Deletes' can be stored up in auto-repeat mode more quickly than they appear on the screen. This will have the effect of 'eating' the text!

Do not hold Delete down. Instead, use single key depressions or Line Delete.

5.4.6 EXIT

Form

CTL/FN 8 - Exit
CTL/FN U - Exit

Use

Exit operates slightly differently, depending on whether the editor was entered from either DEMOS or BASIC. In either case, pressing Exit clears the label line and presents the question:

```
Exit Y/N?
```

answering N aborts the command and returns the cursor to its previous position.

Answering Y when entered from DEMOS exits from the editor with no change to any file. Any alterations are lost.

Answering Y when entered from BASIC converts the text back into a tokenised BASIC program and places it in the BASIC workspace ready for execution, saving, etc.

When Exit is in progress:

```
** Wait
```

is displayed on the bottom line.

5.4.7 FILE START AND END

Form

CTL/FN W - File start
CTL/FN Z - File end

Use

These commands move the cursor to the beginning or end of the file.

File start: positions the cursor on the first character, as if the Editor had just been entered.

File end: places the cursor near the end of the file. The screen displays the last line of the file and puts the cursor on the second text line of the screen.

Notes

Very useful for adding text to a file.

5.4.8 FIND

Form

CTL/FN 2 - Find
CTL/FN O - Find

Use

Find is used to search the file for a specified string. Searching starts at one location point past the cursor position and continues to the end of the file. Text before the cursor is not searched.

After activating Find, the label line is cleared and the message:

Find?

is displayed. Type the string to look for (up to 68 characters) and press ENTER. While searching is in progress the message:

**** Wait**

is displayed. If an exact match for the desired string is found then the cursor is placed on the first character. The case is significant ie capitals will only match capitals and lowercase letters will only match lowercase.

If no match is found, the message:

**** Failed**

is displayed on the bottom line and the cursor returned to its original position.

To search for the same string again, press ENTER in response to the "Find?" prompt. This should not be done for the first search after entering the Editor as the command will be rejected.

Notes

To halt the search, press ESC.

The **** Failed** message disappears after any key depression.

5.4.9 LINE DELETE

Form

CTL/FN 7 - Line Delete
CTL/FN Y - Line Delete

Use

Line delete removes the line on which the cursor is placed.

The cursor must be placed at the beginning of the line.

5.4.10 LINE START AND END

Form

CTL/FN Q - Line Start
CTL/FN T - Line End

Use

These keys move the cursor to the beginning or end of the current line.

Notes

Useful for adding text to the end of a line.

5.4.11 PAGE SCROLL

Form

CTL/FN R - Page Up
CTL/FN C - Page Down

Use

These keys are used to scroll the text page by page.

Page Down will firstly place the cursor at the bottom of the screen being displayed. Further operations will move forwards through the file, six lines at a time.

Similarly, Page Up will initially place the cursor on the top line of the display and then move backwards through the file, six lines at a time.

Notes

Useful for reviewing text.

5.4.12 SAVE

Form

CTL/FN 6 - Save
CTL/FN B - Save

Use

This command is for saving modified text into a file.

After entering the command, the label line is cleared and the following message is displayed:

```
Save Filename ...
```

If you press ENTER, the file is saved using the name that you typed in the original EDIT command. If you entered the Editor from BASIC without specifying a filename, the program will be saved as HUNTER.HBA.

If you type a filename the text will be saved in that file. If you type a .HBA extension, the text will be converted to a **Hunter** BASIC file. If you specify any other extension, it will be saved as ASCII text.

Notes

If there is insufficient room to save the file, DEMOS is re-entered with the error message:

```
Disk Full File length = n blocks
```

where *n* is the number of blocks needed to save all the file currently in the Editor. To save it, create space by deleting any unwanted files (use the STAT command to see how much space there is on the disk) and then type:

SAVE *n filename* ENTER

where *n* is the number that was displayed in the error message.

This will happen even if the Editor was entered from BASIC.

5.4.13 WORD SKIP

Form

CTL/FN A - Left word
CTL/FN F - Right word

Use

These keys move the cursor left and right, pausing only on spaces. Otherwise, they operate exactly as the left and right arrow keys.

Notes

A fast method for gaining position in text.

5.5 Editor Error Messages

EDIT does not use many error messages, as most of its commands are simple and cannot go wrong. Those that are used are as follows:

5.5.1 Beep

You have typed an illegal answer to a prompt or used a command when it is not valid, such as LINE DELETE when the cursor is not at the start of the line.

5.5.2 ** Failed

You have used a FIND command and specified text which has not been found.

5.5.3 ** Wait

Not an error message, this simply indicates that you should wait while the editor carries out a lengthy process such as saving a file.

5.5.4 Disk full File length = *n* blocks

You have tried to SAVE a file when there is not enough space in the **Hunter 2**'s RAM disk for it.

You are taken to the DEMOS File Manager, with the file stored in the TPA. The number *n* is the number of blocks needed to save all the file. To save it, create space by deleting any unwanted files (use the STAT command to see how much space there is on the disk) and then type:

SAVE *n filename* ENTER

where *n* is the number that was displayed in the error message.

This will happen even if the Editor was entered from BASIC.

5.5.5 File R/O File length = *n* blocks

You have tried to SAVE a file which is read-only.

You are taken to the DEMOS File Manager, with the file stored in the TPA. Choose a different filename and type:

SAVE *n filename* ENTER

where *n* is the number displayed in the error message.

5.5.6 Memory overflow!

You have tried to increase the size of the file in the editor beyond the end of the TPA.

SAVE the file, then use a combination of LINE DELETE, SAVE and EDIT to split it into smaller portions, to allow you to continue adding text.

Chapter 6 Introduction to Communications

This chapter provides an introduction to communications and serial interfacing and practical information on selecting the protocol, handshaking and other aspects of the **Hunter 2**'s port.

For details of using communications, see Chapter 7 For more information on the electrical characteristics of the interface, see Part 5. For details of programming with communications, see Parts 3 and 4.

6.1 Introduction

A principle feature of **Hunter 2**'s architecture is its powerful and flexible communications facility.

Hunter 2 can communicate freely with most other types of computer, from microcomputers to mainframes, using a variety of easily selected communication 'Protocols'. These protocols are industry-standard, chosen for compatibility with other systems and are not restricted to **Hunter 2**.

The communications port conforms broadly with the RS232/V24 specification, which is used in virtually every computer system, terminal, printer and modem. This allows **Hunter 2** to 'plug in' to most other systems without modification.

Hunter 2 is configured to act as a 'Data Terminal Equipment' (DTE). By choosing the appropriate protocol, you can make it resemble (partially emulate) any of a number of popular computer terminals, allowing you to use it as a portable computer terminal to access and communicate with other systems.

Hunter 2 can also be used as a self-contained computer, connected directly to suitable peripherals like printers via the communications port. As its port is wired as a DTE, you will need a 'crossed cable' or 'modem simulator' to connect to most peripherals, as they are usually configured as DTEs too.

You can also communicate with other **Hunter 2**s and other Husky computers, either side-by-side on the bench or over thousands of miles by telecommunication channels.

6.1.1 Background Information

You do not need a detailed understanding of data communication to be able to utilise **Hunter 2**'s powerful facilities. You simply need to know how to:

* connect **Hunter 2** to the remote system

* set up **Hunter 2**'s communications to suit the remote system
* use the communications facilities provided

The following background information may however be of interest.

Connecting to a remote system is described briefly in section 6.2 and in more detail in Part 5.

Configuring communications is described in this chapter.

Using communications is described in Chapter 7

Asynchronous Character Handling

Hunter 2's asynchronous serial data communications are handled by a communications software package which is totally separate to the user program and BASIC Interpreter, driven by interrupts from the interface.

The required configuration is selected from 'menus' and operation of the package is virtually invisible.

Interface Power Control

When you turn on, the serial output interface is left turned off to conserve battery life. **Hunter 2** turns it on automatically when a user program attempts to send or receive data, or if the handshake lines become active, waiting approximately 2 seconds to allow the interface to settle down before beginning transmission or reception.

While the port is turning on it may send spurious characters to the remote system, which may cause problems. To avoid this you may turn the port on before connecting to the remote system. For example, by executing a BASIC LPRINT command.

The port remains turned on until you turn the **Hunter 2** off, so further transmissions are not affected in this way. (To save power you may also turn off the port under program control, as described in Parts 3 and 4.)

Buffering

Characters are transferred to and from the port via invisible 'buffers' that effectively disconnect the application program from the outside world. The communications package is driven by interrupts, for maximum efficiency. This ensures a smooth flow of data and correct handling of communications even when **Hunter 2** is busy doing something quite different!

Because the communication package is interrupt driven, the execution of user programs is not affected by communication. However, the user program may slow down if the serial line is busy. **Hunter 2**'s communication buffers are available whenever the machine is powered up, even if the application program is not requesting communication. Incoming messages will be received and held in the buffer, while any outgoing data remaining will continue to be sent down the line. Any protocols selected will continue to be observed.

Data Transmission

A user program transmits data by using BASIC commands such as LPRINT or LOPCHR, or standard CP/M calls. This data is actually written to a 256 character transmit buffer Characters are removed from this buffer by the interrupt-driven communications package as and when the remote system is ready to receive them.

If the buffer becomes full, then the user program (BASIC or CP/M) is held up until the characters or block are sent (or a timeout occurs).

If a timeout or other communications failure is detected, execution of the user program will be suspended until manually overridden. (See section 7.5.)

Data Reception

When data is received from the port, the interrupt-driven communications package places the data in a 256 character wrap-round receive buffer

If the buffer becomes full, the later characters overwrite the earlier characters. This is normally prevented by handshaking or use of a protocol, but may occur if the remote system does not cease transmission when told to.

A user program reads data from the receive buffer by using BASIC commands such as LINCHR and LINPUT or standard CP/M calls. If there is sufficient data in the buffer, then it will be returned immediately. If there is insufficient data, **Hunter 2** will wait for it to arrive (or until a timeout occurs).

Reception of data is completely transparent to the user BASIC program being run by **Hunter 2**. Data may be sent to **Hunter 2** at any time and the communications protocols will function as configured.

If a timeout or other communications failure is detected, execution of the user program will be suspended until manually overridden. (See section 7.5.)

6.2 Connecting to the Serial Port

Hunter 2 uses a standard communications line via a 25 pin D-type connector compatible with the requirements of EIA specification RS-232-C. The connections may be summarised as in the following table:

Hunter 2 is configured as a 'DTE' (Data Terminal Equipment):

Pin	Description
1	Protective Ground
2	Transmitted Data
3	Received Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9	(Not used)
10	5v Power Out
11	(Not used)
12	(Not used)
13	(Not used)
14	(Not used)
15	Transmit Clock
16	(Not used)
17	Receive Clock
18	(Not used)
19	(Not used)
20	Data Terminal Ready
21	(Not used)
22	Ring Indicator
23	(Not used)
24	(Not used)
25	(Not used)

6.3 Configuring the Port

6.3.1 Introduction

Hunter 2's communications format is **software controlled.**

If **Hunter 2** fails to communicate, always check the port initialisation for the correct format.

6.3.2 Using the On-screen Menu

To examine, initialise, or alter, **Hunter 2**'s communications parameters proceed as follows:

From the File Manager type:
CTL/FN 4
or
COMS ENTER

If using an application, select the 'Initialise communications' option.

This displays the current transmission parameters, in a screen menu format like this:

```
Transmission Parameters
Rate-1800  Prtcl-none      Pty-none
CTS-n DTR-n LF-n Echo-y T/O-no Null- 0
press ENTER if acceptable.
```

If you want to modify a parameter, move the cursor to it by using the ⊕ and ⊖ keys, then press the ↑ or ↓ keys to cycle through the list of allowed options.

When you have configured transmission as you wish, press ENTER to select the receiving parameters screen:


```
Receiving Parameters
Rate-1200  Prtcl-none      Pty-none
RTS-off  DSR-n DCD-n  T/O-no Serig-off
press ENTER if acceptable
```

If you wish to modify any of the parameters, use the cursor keys as described previously.

When the configuration is as you wish, press ENTER.

If you simply want to examine the parameters, press ENTER without using the cursor keys.

NOTE: If the **Hunter 2** has been totally reset for any reason, the parameters shown by the menu will be different from those you expect. You will still be able to configure communications as you want by using the cursor keys.

The configuration options are described in section 6.4, in alphabetical order.

6.3.3 Configuring the Port from Programs

You may also configure the port from user programs, by writing to flags held in memory. See Part 3 (BASIC) and Part 4 (assembly language) for more details.

6.4 The Port Set-up Options

The various communications port set-up menu options will now be described, in alphabetical order Once set up correctly, these are preserved through normal usage.

6.4.1 BAUD RATE

Use

To set the transmit and receive baud rates to suit the connected device.

Notes

The available baud rates are:

75	1200
110	1800
150	2400
300	4800
600	CLK

The CLK feature is to provide for external clocking when using synchronous protocols. It is only available when the BISYSNC.SYS protocol is installed.

6.4.2 CTS

Use

To control **Hunter 2**'s response to the CTS (Clear To Send) input (pin 5) during transmission.

Notes

Setting	Result
y	**Hunter 2** will continue transmitting while CTS is active (positive). CTS is tested at the start of each character in asynchronous transmission or the start of each block in synchronous transmission. If CTS goes inactive part way through transmitting a character, that character will be completed before transmission stops.
n	CTS is ignored.

An internal pull-up resistor is fitted to CTS to allow transmission when the host is not driving CTS.

6.4.3 DCD

Use

To control **Hunter 2**'s reaction to the DCD (Data Carrier Detect) input (pin 9), while receiving data.

Notes

Setting	Result
y	**Hunter 2** will receive data normally only while DCD is active (+ve).
n	DCD is ignored.

6.4.4 DSR

Use

To control **Hunter 2**'s reaction to the DSR (Data Set Ready) input (pin 6), while transmitting or receiving.

Notes

The connected device (eg modem) can use this DSR to tell **Hunter 2** when it is ready to operate.

Setting	Result
y	**Hunter 2** will transmit or receive only while DSR is active (+ve).
n	DSR is ignored.

6.4.5 DTR

Use

To control production of the DTR (Data Terminal Ready) output (on pin 20) while transmitting or receiving.

Notes

This signal is provided to allow **Hunter 2** to use auto-answer modems that require DTR to be present before generating the carrier

Setting	Result
y	DTR goes active (+ve) if **Hunter 2** is ready to receive or transmit.
n	DTR goes active if **Hunter 2** is ready to transmit.

DTR can also be controlled by writing to the communications power supply control port, INVCON DTR is active when the invertor is on.

6.4.6 ECHO

Use

To control whether characters will be displayed on the screen as they are transmitted.

Notes

Data transmission and reception are not affected and the displayed characters are not passed to user programs or entered in the receive buffer.

By setting ECHO to 'y', you can mimic full duplex operation with a half duplex remote system (although it does not of course indicate that the remote system has received the characters correctly). If the remote system is operating in full duplex mode, each character will be displayed twice!

Setting	Result
y	Characters are echoed as they are transmitted.
n	Characters are not echoed.

6.4.7 LF

Use

To control whether **Hunter 2** inserts a LF character (decimal 10, hexadecimal 0AH) after each CR (decimal 13, hexadecimal 0DH) when transmitting.

Notes

This option is mainly provided for use with printers or other remote systems that require a CR and LF to start a new line.

Setting	Result
y	A LF will be inserted after each CR transmitted, even if there is already a LF in the text.
n	Extra LFs will not be inserted, although any LFs in the data will be transmitted as normal.

NOTE: When sending binary data (such as a machine code program), set LF to 'n' to avoid corrupting the data.

6.4.8 NULL

Use

To control whether **Hunter 2** inserts NULL characters after each CR, when transmitting.

Notes

This option is mainly provided for use with printers that take a significant time to print a CR character and do not have handshaking or buffering. NULLs are not printed, they simply give the printer time to perform the CR function.

Setting	Result
0	No NULLs are added.
2	2 NULLs are added.
5	5 NULLs are added.
10	10 NULLs are added.
20	20 NULLs are added.

If LF is set to 'y', any NULLs will be sent after the CR LF sequence.

NOTE: If transmitting binary files, set NULL to 0 to avoid corrupting the data.

6.4.9 PRTCL (Protocol)

Use

To select the protocol to use when transmitting or receiving data.

Notes

The protocols available on **Hunter 2** are:

NONE
XON/XOFF
ETX/ACK
ACK/NAK
SYSTIME
IBM 2780 (optional)

See sections 6.6 and 6.7 for further details of these protocols.

6.4.10 PTY (Parity)

Use

To select the parity to use when transmitting data and the type of parity checking to use when receiving data.

Notes

When transmitting, PTY has the following significance:

Setting	Result
None	bit 8 is reset, ie logical 0.
Odd	bit 8 is set or reset to cause an odd number of logical 1's in the data bits.
Even	bit 8 is set or reset to cause an even number of logical 1's in the data bits.
8bit	All eight bits are transmitted with no modification. This may be used to transfer binary information.
8+od	8 data bits plus a ninth parity bit, set or reset to cause an odd number of 1's in the data bits
8+ev	8 data bits plus a ninth parity bit, set or reset to cause an even number of 1's in the data bits

When receiving, PTY has the following significance:

Setting	Result
None	Parity bit is ignored and set to logical 0.
Odd	The data is checked for an odd number of logical 1's (including the parity bit). If the check succeeds, the parity bit is set to 0 and the character is returned to the calling program. If the check fails, the character is received as 255 decimal (0FFH) and a parity error is flagged
Even	The data is checked for an even number of logical 1's (including the parity bit). If the check succeeds, the parity bit is set to 0 and the character is returned to the calling program. If the check fails then the character is received as 255 decimal (0FFH) and a parity error is flagged
8-bit	All eight bits of data are passed to the calling program.

Notes: The parity fail character (0FFH) is displayed as ← on the screen.
On versions before 9G08, 8+od and 8+ev are not implemented.

6.4.11 RTS

Use

To control production of the RTS (Request To Send) output (on pin 4) while transmitting or receiving data.

Notes

OFF	The RTS line is permanently active (+ve).
HOLD	The RTS line is normally active (+ve) and only goes inactive (-ve) when **Hunter 2**'s receive buffer is full. This allows **Hunter 2** to hold up a transmitting device (by lowering the RTS line) until it has removed enough characters from the receive buffer to allow the transmitting device to send more data (by raising the RTS line again). This mode can be used when communicating **Hunter 2** to **Hunter 2** via a crossed cable. The RTS line would normally be connected to the transmitting device's CTS line. If a synchronous protocol has been selected, RTS should not be set to HOLD.
TRUE	This mode of RTS is meant for handshaking with a MODEM or similar device. When **Hunter 2** wishes to transmit data, it first of all raises RTS from inactive (-ve) to active (+ve). Then **Hunter 2** waits for a reply on the CTS line (if enabled) before sending the

data in the transmit buffer RTS remains active until all the data has been sent and the buffer is empty, then **Hunter 2** returns RTS to the inactive state (-ve).

6.4.12 RX T/O (Receive Timeout)

Use

To set the timeout period for data reception.

Notes

The timeout is initiated by a request for serial input data through a BASIC 'LINPUT' or similar statement. If the calling program fails to read a character from the receive buffer within the specified time limit (because it is empty), a communications error will result.

Reception of any character, or noise, will reset the timeout.

If no attempt is made to read the input communications buffer, this timeout will not occur

Setting	Result
0	No timeout
10	Timeout after 10 seconds
20	Timeout after 20 seconds
30	Timeout after 30 seconds
40	Timeout after 40 seconds
50	Timeout after 50 seconds
60	Timeout after 60 seconds

If a protocol is being used which has its own timeouts (ACK/NAK, or SYSTIME), disable the RX timeout.

6.4.13 SERIG

Use

To filter a particular character from received data.

Notes

This tells **Hunter 2** to remove a particular character from serial input ie to receive it normally but not place it in the receive buffer, so the character is never passed to the calling program.

This option is provided for use with computer systems that provide abundant spurious characters (eg XON or DEL) as part of data streams. Since filtering these unwanted characters can be inconvenient in user programs, SERIG provides a useful solution.

Setting	Result
0-127	Filter the specified ASCII code form received data.
off	Do not filter any characters.

To set SERIG to 'off', increment past 127 or decrement past 0.

If you are receiving 8 bit data, ensure that SERIG is set to 'off'

6.4.14 TX T/O (Transmit Timeout)

Use

To set the timeout period for data transmission.

Notes

The timeout is initiated by a request for serial output through a BASIC 'LPRINT' or similar statement. If the calling program fails to put a character in the transmit buffer (because it is full) within a specified time limit, a communications error will occur

If no attempt is made to transmit data, this timeout will not occur.

Setting	Result
0	No timeout
10	Timeout after 10 seconds
20	Timeout after 20 seconds
30	Timeout after 30 seconds
40	Timeout after 40 seconds
50	Timeout after 50 seconds

60 Timeout after 60 seconds

If a protocol is being used which has its own timeouts (ACK/NAK, or SYSTIME), disable the TX timeout.

6.5 Handshaking

The communications port includes a number of handshaking lines. These are usually used when operating with a modem. The lines conform to the standard RS232 specification for use with modems, and may be configured to suit particular requirements. In general, a straight 25-way cable will operate with most modems.

Handshaking may also be used when communicating with slow devices, such as printers, or to allow the **Hunter 2** to suspend data transmissions to it.

The individual handshaking lines are described in turn in section 6.4, with the exception of Ring Indicator (RI). RI is provided to allow **Hunter 2** to be switched on from an external source, such as an auto-answer modem.

NOTE: Many of the handshaking lines are software controlled. If **Hunter 2** fails to communicate, check that handshaking is configured correctly, as described in section 6.3.

6.6 Asynchronous Protocols

6.6.1 Introduction

In this section, 'host' means the device communicating with **Hunter 2**, whether it be computer, printer, modem or otherwise.

These protocols may be used by BASIC application programs, user assembly language code, CP/M programs or directly, in 'Terminal Emulation' mode. They may only be used for transferring ASCII data, as the transmission of 8 bit data could cause some data to be interpreted as protocol control characters.

Protocols provide 'software handshaking', allowing you to communicate over simple 3-wire links (earth, Tx and Rx lines)

6.6.2 NONE

Reception

As its name implies, this selection does nothing to control the incoming data. It behaves as a simple Teletype. Characters are placed into the reception buffer and a calling program may read them out asynchronously. Reception would normally be controlled by hardware handshaking.

Transmission

Each character is placed in the transmission buffer when the calling program passes it (if there is room in the buffer). Transmission would normally be controlled by hardware handshaking.

6.6.3 XON/XOFF (DC1/DC3)

Reception

If an incoming character causes the receive buffer to have 90 or more characters then an XOFF character (DC3, 13H or Control S) is transmitted, which should prevent the host sending further characters to **Hunter 2**. When the buffer is reduced to 10 pending characters, an XON (DC1, 11H or Control Q) is transmitted to re-enable the host to **Hunter 2** transmission. Subsequent XONs or XOFFs are never transmitted without an intervening XOFF or XON respectively

Transmission

The input data line is also examined for XON or XOFF characters.

If an XOFF or stream of XOFFs are received then transmission is stopped as soon as the character being transmitted is finished. To restart transmission, the transmitter must send another XON.

This protocol procedure strips XON and XOFF Control characters, thus returning only valid data.

By the very nature of this protocol, indeterminate handshaking situations may occur if any conditions are altered during transmission.

The transmitting system does not need to send an XON at the start of transmission.

After the host sends an XOFF, **Hunter 2** may send up to two more characters, depending on data link speed. This is due to the time taken to receive the XOFF and the possibility of just missing the start of the next character

6.6.4 ETX/ACK

Reception

In this protocol, the host transmits data in blocks. Each block is terminated by a single ETX character (03H or Control C) and must not be more than 130 characters. The host waits after transmitting a block, for **Hunter 2** to send an ACK character (06H or Control F)

The ETX character at the end of the block is stripped from the data and returned to the calling program as CR (0DH or Control M).

Hunter 2 only sends an ACK acknowledgement when the block has been read by the calling program.

Transmission

Hunter 2 sends an ETX character after any transmitted carriage return. Transmission is then halted until an ACK (06H or Control F) is received from the host. This enables individual data blocks to be sent and an acknowledgement awaited. If transmission is to another **Hunter 2**, the blocks should not exceed 130 characters or the input buffer may be exceeded. If an ACK is not received within thirty seconds then a communications error is generated.

ACKs are not returned to the user program.

By the very nature of this protocol, indeterminate handshaking situations may occur if any conditions are altered during transmission.

6.6.5 ACK/NAK

Reception

In this protocol **Hunter 2** will receive a complete block of data, terminated with a CR (0DH or Control M). It will then check for parity errors (if parity is not set to 'none'). If there is a parity error, a NAK (15H or Control U) is sent to the host and the data block is ignored.

If the block is error-free then the received block is sent character by character to BASIC. When the carriage return is sent to BASIC an ACK (06H or Control F) is sent to the host to indicate successful data block reception.

Transmission

Data is placed in the transmission buffer until a CR is received from the calling program, then the entire block is transmitted. **Hunter 2** then waits for an acknowledgement. If it is an ACK, the next data block is assembled in the buffer and sent. If it is a NAK (15H or Control U), this signals a data error and the block is sent again. If nothing is received after three seconds, a NAK is assumed and the block is sent again. A block will be sent a maximum of four times; if an ACK is not received after the fourth block, a communications failure will be reported.

6.6.6 SYSTIME

This protocol embodies a checksum in addition to parity checks. Data is sent in the form of messages which are subdivided into one or more fixed length blocks. Each block is individually checked and acknowledged before the next blocks are sent.

This protocol provides the only 'secure' communications method available in asynchronous mode.

Data Blocks

Each block consists of:

1 start character
64 data characters
3 checksum characters
1 stop character

The purpose of each character group is as follows:

Start Character	The first block of a message uses SOH (01H). Further blocks use STX (02H) as the start character
Data Character	The data characters may consist of any 7 bit pattern with the exception of the handshake characters: ACK (06H), NAK (15H) and WACK (13H). If there are less than 64 characters for the data block, then the remainder of the block is filled with NULLs (00H). NULL characters are ignored on reception.
Checksum	The checksum is the remainder when all the data characters are added together with their parity bits set to 0 and the result is divided by 256. This number formed is transmitted in ASCII as three decimal digits. Leading zeros are not suppressed. The most significant digit is sent first.
Stop Character	The last block of a message uses EOT (04H) as the stop character, while previous blocks use ETX (03H).

Handshaking

Each block of data must be acknowledged by the receiving device, as follows.

If the block is received successfully and the receiver is ready to receive another block, it returns an ACK (06H).

If the transfer is deemed unsuccessful for any of the following reasons then a failure is indicated by returning a NAK (15H):

- parity fault on any character (if parity checking is enabled)
- incorrect start or end characters
- wrong number of characters
- checksum error
- timeout (waiting for characters for 4 secs)

If the block is received successfully but the receiver is not ready for further data, it returns a 'wait acknowledge character'· WACK (13H). This delays the transmitter's timeout for three seconds. To delay transmission further, the receiver sends a further WACK every 2 seconds. When the receiver is ready for another block, it sends an ACK.

6.7 Synchronous Protocols

To provide the fast, high integrity communication required by most mainframe computers, **Hunter 2** has a synchronous data communication capability.

Communication takes place using synchronous transmission of data blocks. These blocks are checked using check characters and acknowledged as appropriate using particular protocols.

Currently available as a factory option is the IBM 2780/3780 communication protocol providing bisynchronous communications.

6.7.1 IBM 2780/3780 on Hunter 2

Background

The IBM 2780 is a very popular data transmission terminal used for remote job entry. The transmission protocol developed for this device has been widely implemented on other mainframes to the extent of becoming a de facto industry standard. **Hunter 2**'s implementation mimics the IBM 2780 protocol to provide well proven, fast and extremely reliable data transfer.

The IBM 2780 is used for batch entry by use of 'job cards'. **Hunter 2** is easily configured to simulate job cards, providing all the processing facilities normally available on a remote terminal.

Hunter 2 is the most advanced portable computer to offer a comprehensive implementation of IBM 2780 in true synchronous mode.

Advantages

This well proven and sophisticated protocol gives **Hunter 2** reliable high speed communications even over fairly poor data links such as the public switched telephone network (PSTN), at speeds as high as 2400 baud.

Use of this standard provides access to a wide range of mainframe computer systems, directly and without need for supporting equipment. Data files can be exchanged between **Hunter 2** and mainframes with great flexibility, convenience and dependability.

- Synchronous transmission using Bisynchronous protocol
- Character set - EBCDIC
- Full CRC generation, error handling and flow control.
- Modem handshake lines available.
- Internal or external clocking options.

* May be used over the public switched telephone network or private lines.
* Records terminated using the EM character, providing optimum throughput.
* "Wait acknowledge" is fully implemented.
* Protocol use is completely transparent to user programs.
* Data Rates - up to 9600 Hz with external clocking, up to 4800 Hz using internal clocking

The internal self-synchronising feature enables **Hunter 2** to **Hunter 2** communications over standard 300 baud asynchronous modems.

6.7.2 Synchronous Communication

Synchronous communication transfers data as a continuous stream of data characters, without the need for start and stop bits as used in asynchronous communication. This maximises the data transfer rate for a given channel bandwidth. For the receiver to be able to use the data two levels of synchronisation need to take place:

i) Bit Synchronisation	This allows the receiver to save the state of each bit correctly, preferably sampled at the centre of each bit.
ii) Character Synchronisation	Each block of eight bits which represent a character needs to be identified from within the data stream.

Bit Synchronisation

Hunter 2 uses two techniques to achieve bit synchronisation.

External Clocking allows both transmit and receive to be synchronised using two independent clocking signals. These signals are generated by either a modem or modem eliminator Selecting 'CLK' as the speed in the transmit or receive menus selects this mode.

Two pins are provided for the clock signals. Pin 15 provides the transmit clock and pin 17 the receive clock.

On transmit, a rising edge (-12V to +12V) causes a bit to change on the transmit output (pin 2). On receive, a falling edge (+12V to -12V) causes the state of the receive input (pin 3) to be sampled.

It is important that maximum clock speed on **Hunter 2** should not exceed 9600 Hz (giving a data rate of 9600 baud). The minimum speed is approximately 100Hz.

Exceeding this clock rate may cause unpredictable results.

Self clocking is achieved by selecting from the menu the desired transmit and receive baud rates in the range 50-4800 baud, as for asynchronous. On transmit, the data is sent out by using internal timing, with no reference to any external clock source.

When receiving data, **Hunter 2** derives its internal clocking by timing the rising edges (-12V to +12V) on the receive data line.

Self clocking is useful for **Hunter 2 - Hunter 2** communications. It may be used over asynchronous modems to provide both increased throughput and data integrity.

Character Synchronisation

To be able to extract eight bit data characters, **Hunter 2** has to synchronise itself to the incoming characters correctly This is done by detecting special synchronising characters called SYN characters (32H). **Hunter 2** considers synchronisation to have been achieved after receiving two contiguous SYN characters.

After receiving the two synchronising characters each group of eight bits are then considered to be a character

Synchronisation is considered to be lost if a line turnaround character (0FFH) is received. This is equivalent to an open line.

Before sending the synchronising characters, **Hunter 2** transmits a pad character (0AAH) to assist bit synchronisation.

Every data block, whether it consists of one or 500 characters, must start with a pad character and two SYN characters, and terminate with line turnaround.

6.7.3 Binary Synchronous Communications (BSC)

The information carrier for 2780 is the binary synchronous communications (BSC) procedure. The character set used is EBCDIC (Extended Binary Coded Decimal Interchange Code). The following is a description of the subset of BSC used for **Hunter 2** 2780 emulation.

Hunter 2 generally operates using the USASCII (United States of America Standard Code for Information Interchange) character standard, so that characters transmitted by or received by the calling program will be in ASCII. The 2780 emulation automatically converts ASCII to EBCDIC (on transmission) and EBCDIC to ASCII (on reception). The conversions used are listed in Appendix IV.

The code used on the transmission channel is transparent to the calling program, and need not concern the programmer.

Text Blocking

BSC defines general structural procedures for the blocking of information. Several control characters are used for the control of message blocks.

A message consists of one or more blocks of text data. The start of text character (STX) is used immediately preceding each block of data. Each data block but the last is terminated by an end of transmission block (ETB) character or an intermediate text block (ITB) character. The last data block ends with an end of text (ETX) character.

Error Checking

Each block of data is error checked by the receiver by a cyclic redundancy check (CRC). After each transmission the receiver normally replies with ACK0 or ACK1 - data accepted continue sending; or with NAK - data not accepted (eg due to a data transmission error), retransmit the previous block.

CRC-16

Hunter 2 uses the accepted error checking method of EBCDIC 2780. The cyclic redundancy check is a division at both the transmitting and receiving stations using the numeric value of the message as a dividend, divided by a constant. The quotient is discarded and the remainder is used as the check character

This is transmitted immediately following an ITB, ETB or ETX.

The receiver checks this value and finds no error if they are equal.

CRC-16 uses the polynomial $(x^{16}+x^{15}+x^2+1)$ as the divisor constant. The 16 bit remainder is sent as two eight bit characters. This is also sometimes referred to as the BCC (Block Check Character).

Line Bid

For either **Hunter 2** or base computer to transmit a message, the line must first be seized. This requires sending an enquiry (ENQ) character. A positive acknowledgement permits the start of a message.

6.7.4 Link Control Characters

The data link is controlled by the use of the following control characters:

SYN (32H) - Synchronous Idle

This character is used to establish synchronisation. Two contiguous SYNs are required for synchronisation. They may also be used as pad characters, although never used as such by **Hunter 2.** SYNs will be ignored after synchronisation has been established except as part of CRC.

STX (02H) - Start of Text

This character is used as the start of text preceding all text blocks. It is not part of the CRC accumulation.

ETB (26H) - End of Transmission Block

The ETB character indicates the end of a block of data characters. A block check character is sent immediately following an ETB. A reply is required after an ETB (ACK0, NAK, etc).

ITB (1FH) - End of Intermediate Transmission Block

ITB is also called IUS or US (Unit Separator). This enhances the throughput due to not having to go through a line turnaround. The CRC accumulation is reset after an ITB. It is not necessary to send an STX after the ITB.

If the error check fails then a NAK is sent at the end of the transmitted message, ie after the ETB or ETX. The whole of the message is retransmitted in the event of a NAK being the reply to a complete message block.

ETX (03H) - End of Text

ETX is transmitted after a block of characters started by an STX. The CRC is sent immediately after an ETX. A reply is required to indicate the receiver status.

EOT (37H) - End of Transmission

This character terminates a message transmission containing a number of blocks. It is also used to indicate a system malfunction.

ENQ (2DH) - Enquiry

ENQ is used as the line bid to start a transmission sequence. ENQ is also used to obtain a repeat transmission of a response, in the case of a garbled response.

ACK0 (10H,70H) - Affirmative Acknowledgement

This is one of two positive acknowledgements to error checked message buffers. This informs the transmitting station that the message was received satisfactorily. This acknowledgement is also used to confirm a request for the line. ACK0 alternates with ACK1 positive acknowledge.

Alternating Affirmative Acknowledgement

Alternating ACK0 and ACK1 provides sequential checking of replies. This ensures that the blocks are replied to correctly. ACK0 is used as the response to a line bid.

Note that ACK0 and ACK1 are both two-character sequences.

ACK1 (10H, 61H) - Positive Acknowledge

This is the alternate acknowledge. It is used with ACK0 to respond correctly to incoming messages.

WACK (10H, 1CH) - Wait Positive Acknowledge

A receiving station that has correctly received a message but is not yet in a condition to use the data, can send a WACK. It is a positive acknowledge. The transmitting station should then respond with an ENQ. WACK is also a two character sequence. WACKs may be sent indefinitely at 3 second intervals to stop further transmission.

NAK (15H) - Negative Acknowledge

This character informs the transmitting station that a message was received in error It causes retransmission of the erroneous message.

DLE (10H) - Data Link Escape

Data link escape is used to provide various line control characters, eg WACK, ACK0, ACK1 and RVI.

RVI (10H, 7CH) - Reverse Interrupt

Reverse interrupt is used as a positive acknowledgement (in place of ACK0 or ACK1). However, it requests termination of the current message, due to a high priority message needing to be sent from the receiving station.

Hunter 2 will never generate an RVI but will respond to one.

DLE EOT (10H,04H) - Disconnect Sequence

This sequence informs the receiver that the transmitter is shutting down.

IRS (1EH) - Intermediate Record Separator

This character is used as a delimiter in 3780 configurations. This character is returned to the calling program as CR (0CH).

6.7.5 Transparency

(Transparency is not available on versions earlier than 9G08.)

Introduction

Transparency allows transmission of 8-bit data, such as binary data, floating point numbers and machine code programs.

Transparency is accomplished by preceding the line control characters with a Data-Link-Escape code, DLE (10H).

The following code sequences are used when in transparent mode:

DLE STX. Indicates transparent mode for the following record of data.

DLE ETB: Terminates a block of transparent data followed by the CRC and calls for a reply, (either ACK0, ACK1, WACK, NAK). Returns the data link to normal mode.

DLE ETX: Terminates transparent data mode, followed by the CRC and calls for a reply. Returns the data link to normal mode.

DLE ITB: Terminates an intermediate block of transparent data, ie when a block contains multiple records. The intermediate CRC follows the DLE ITB. This sequence does not call for a reply.

DLE ENQ: An indication to disregard this block of transparent data and return the data link to normal mode.

DLE DLE: This is the sequence used to transmit a 10H byte, the preceding DLE is stripped out of the received data by the receiving station.

DLE SYN: This is used to maintain synchronisation or as a time fill sequence, but cannot be used to initiate synchronism.

Notes

When transparency is in effect, the ASCII character set is used since this will allow 256 codes. Transparent mode is less efficient than normal mode, (where normal mode is defined as the mode which is used when transparency is not in effect), because the above line control characters must be preceded by DLE, and the data 10H must itself be preceded by DLE. Also, space compression cannot be used. It is therefore advisable to only use transparency when it is essential.

The initial PAD, SYN, SYN, TURNAROUND characters to establish a data link and synchronisation are not preceded by DLE.

Reply codes (ACK0, ACK1, NAK, WACK) are transmitted in normal mode.

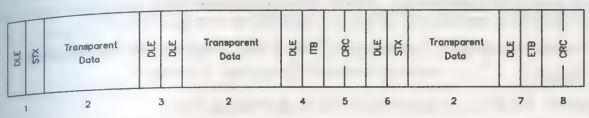
Error-checking is accomplished using CRC-16 as is the case in non-transparent mode.

When the data to be transmitted corresponds to DLE, as stated above, a preceding DLE is inserted into the record. However, this preceding DLE does not form part of the record count nor part of the CRC.

If several DLE data codes are to be transmitted within a record, then the user must ensure that the receive buffer is capable of storing more characters than one would expect in the record.

Example

A typical 2780 transparent data block looks like this:



KEY:

1) The initial DLE STX at the start of the block.
2) Transparent data.
3) The data DLE is preceded by DLE.
4) End of intermediate block control characters.
5) The intermediate CRC 16.
6) This DLE STX signifies the following record is transparent.
7) The end of block control characters.
8) The CRC-16 for the last intermediate block transmitted. It is at this point that a reply is expected by the transmitting station.

A 3780 transparent data block is similar to the above except that fields 4, 5 and 6 do not appear and thus more data may be packed into a data block.

Using Transparent Mode

The operating system will automatically recognise the reception of transparent data due to the transparent data link character sequence DLE STX.

It will only transmit data in transparent mode if RAM location TRANTX (DFBCH, 57276 decimal) is set to 1. If TRANTX = 0 then normal mode is used.

The SETSYNC8.HBA program can be used to configure the IBM2780 protocol. This allows you to select transparent mode via a menu. SETSYNC8.HBA is listed in section 6.7.8.

SETSYNC8 uses:

```
A = ARG(38):A=CALL(5)
```

This calls a routine in the operating system to initialise the communications port and associated parameters. This is necessary when using transparent mode so that the initial handshake buffer is initialised to contain DLE STX.

To select transparent transmission from a user program, use the following:

In BASIC:

```
POKE 57276,1:A=ARG(38):A=CALL(5)
```

In assembly language:

```
TRANTX    EQU       0DFBCH
;
          LD        A,1
          LD        (TRANTX),A          ;Set transparency mode
          LD        C,26H               ;Communication Initialisation
          CALL      5                   ;Call number
```

6.7.6 Message Transmission

The following figures illustrate various message sequences and error handling.

NOTE: TX = Transmitting station. RX = Receiving station.

Figure 6.1: Normal Message

```
                ODD         EVEN        ODD
     E  S     E  S     E  S     E  S     E  E
TX:  N  T(TEXT)T  T(TEXT)T  T(TEXT)T  T  O
     Q  X     B  X     B  X     B  T

       A           A           A           A
RX:    C           C           C           C
       K           K           K           K
       0           1           0           1
```


Figure 6.2: Unanswered Line Bid (Error 01)

```
     E (3 second  E (3 second  E (number      E
TX:  N   bid      N   bid      N   of         O
     Q  timeout)  Q  timeout)  Q retries)..... T
```

NOTE: Hunter 2 will retry for a total of 10 ENQs. Many mainframe systems never give up.

Figure 6.3: Accepted Re-transmissions

```
                ODD         ODD         EVEN
     E  S     E  S     E  S     E  S     E  E
TX:  N  T(TEXT1)T  T(TEXT1)T  T(TEXT2)T  O
     Q  X     B  X     B  X     B  T

       A           N           A           A
RX:    C           A           C           C
       K           K           K           K
       0                       1           0
```

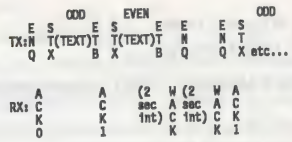
Figure 6.4: Re-transmission Rejected (Error 03)

```
                ODD         ODD         ODD
     E  S     E  S     E  S     E  S     E  E
TX:  N  T(TEXT1)T  T(TEXT1)T  T(TEXT1)T  O
     Q  X     B  X     B  X     B  T

       A           N           N           N
RX:    C           A           A           A
       K           K           K           K
       0
```

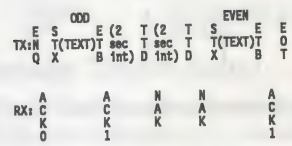
NOTE: The number of retries can vary. **Hunter 2** will retry 10 times, but some mainframe systems will retry indefinitely

Figure 6.5 Transmission Delay (Receiver Initiated)



NOTE: Hunter 2 does not count WACK-ENQ sequences. The receiver buffer may be cleared.

Figure 6.6 Transmission Delay (Transmitter Initiated)



NOTE: Hunter 2 does not count TTD/NAK sequences or generate TTDs.

Figure 6.7: STX Lost and Data Ignored

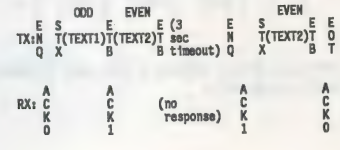


Figure 6.8: Incorrect Positive Acknowledgement (Error 03)

NOTE: Hunter 2 will send 10 ENQs before giving up unless programmed to perform infinite retries (using SETSYNC8).

Figure 6.9: Data Link Abort on no Response (Error 02)

NOTE: Hunter 2 sends out 10 ENQs.

Figure 6.10 Data Link Stalemate

NOTE: Failure to receive further messages could be timed out by the user program.

6.7.7 Hunter 2 2780 Implementation Features

The previous section describes the general principles of operations of the IBM 2780 protocol. **Hunter 2**'s implementation has a number of particular features which may be important to users.

Buffer Operation

To implement this synchronous protocol it is necessary to use buffering on transmit and receive so that error checking and re-transmission can be performed.

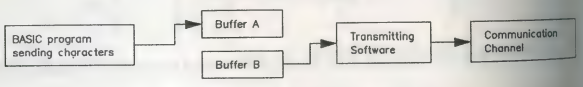
To enhance throughput there are two buffers each for transmit and receive, the operation of which is described below.

Transmit Buffer

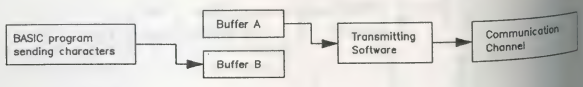
While the maximum buffer size required by IBM 2780 is 512 characters, some systems are only configured for 256 characters. For this reason **Hunter 2** will default to transmitting 200 characters for ease of interfacing.

On transmit, while one buffer is being filled from the user program the other is being sent. When the new buffer is full, the buffers effectively swap over and the cycle continues.

First buffer A fills up while buffer B transmits:



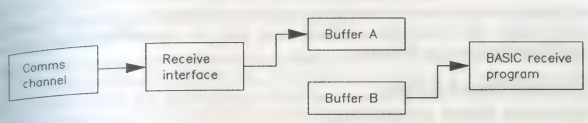
Then, buffer B fills up while buffer A transmits:



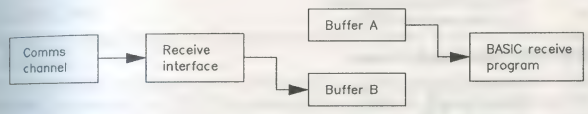
Receiving buffer

The maximum number of characters that **Hunter 2** can buffer (including control characters) is 512.

First, buffer A fills, while buffer B empties:



Then, buffer B fills while buffer A empties:



As with the transmit buffer arrangement, one buffer is being filled from the communications line while the other is being emptied by the user program.

Data Format

Hunter 2 can accommodate 2780 protocol with fixed or variable block lengths and the 3780 protocol.

It is possible to configure the bisynchronous protocol to provide very flexible interfacing for a variety of different host machines.

Standard configurations can be made for:

- 2780 with variable record length
- 2780 with fixed record length
- 3780

The block sizes and record sizes can be adjusted for compatibility and maximum throughput.

Other features which are available for special purposes are:

* Intermediate STX
* Single record blocks
* Space compression
* Transparency

The multidrop feature is not implemented.

All these features are discussed in detail later Although they may all be selected individually, the resulting protocol may not be compatible. The standard configurations are as follows:

2780 with Variable Records

(This is the default configuration)

* Record length = 80
* Block length = 200
* Multiple records per block
* Record Padding disabled
* No intermediate STX

2780 with Fixed Length Records

* Record length = 80
* Block length = 200
* Multiple records per block
* Record padding enabled
* Intermediate STX enabled

3780

* Record length = 80
* Block length = 512
* Multiple records per block
* Record padding disabled
* No intermediate STX

Space compression is normally used only in 3780 installations.

Hunter 2 Card Format

Virtually all **Hunter 2** data is delimited by CR (carriage return).

Blocks of data loaded by LINPUT statements are terminated by CR, as are lines of printout. The most straightforward method to create blocks is to treat data between CRs as cards of data.

Transmitting Cards

On transmit, if a CR is detected in the data, the communication software will treat it as a record end. The action taken depends upon the amount of room left in the current buffer being prepared for transmission.

For further details on end of record action see section 6.7.8.

Sending the Last Card

To maintain compatibility with existing transmission programs it is essential they should operate without modification. This leaves a problem of how to send the final message buffer if it has not been filled up. If greater than 5 seconds elapses between characters sent from the user program, then the end of a complete transmission sequence is assumed and the current block is terminated by ETX CRC. After this has been acknowledged, EOT (end of transmission) is sent to close down the line.

Receiving Cards

The computer generating data for transmission to **Hunter 2** will also block data into records.

To provide user programs with "CR" delimiters, any end of record character received is converted to CR. The character used for end of record depends on the configuration in use. The sequences IUS CRC and IRS will always return CR.

This means that **Hunter 2s** will communicate directly when connected back to back.

BSC receiving stations use ESC sequences for special functions like skipping line. These sequences are ESC*n* where *n* can be a number of printable characters. To avoid confusion, **Hunter 2** will ignore received ESCs and their associated following character

Character Restrictions

Operating system versions earlier than 9G08 do not provide any of the transparency features of BSC. It is therefore not possible on these earlier versions to send any of the following characters:

STX - 02H
ETX - 03H
ETB - 17H
EDT - 04H
SYN - 16H
NAK - 15H
DLE - 10H
US - 1FH
ESC - 1BH

If any of these characters are sent (ie included by the user program in the data to send) then they may either be lost or cause disruption to the transmission channel.

Block Integrity Checks

(This facility is not available for versions earlier than 9G08.)

The operating system maintains checks on the block integrity, in both normal and transparent modes of operation. The following RAM locations may be examined at any time to indicate the present integrity status, or the integrity status for the most recent data link.

NAME	ADDRESS (hex,decimal)	FUNCTION
TXGOODS	0DFB4H, 57268	lsb number of good transmitted blocks
TXGOODS+1	0DFB5H, 57269	msb number of good transmitted blocks
TXBADS	0DFB6H, 57270	lsb number of bad transmitted blocks
TXBADS+1	0DFB7H, 57271	msb number of bad transmitted blocks
RXGOODS	0DFB8H, 57272	lsb number of good received blocks
RXGOODS+1	0DFB9H, 57273	msb number of good received blocks
RXBADS	0DFBAH, 57274	lsb number of bad received blocks
RXBADS+1	0DFBBH, 57275	msb number of bad received blocks

The status bytes given above are maintained by the operating system in the following manner:

TXGOODS:
* Initialised to 0 when the initial ENQ to establish a data link is transmitted.
* Incremented when the appropriate ACK0 or ACK1 is received.

TXBADS:
* Initialise to 0 when the initial ENQ to establish a data link is transmitted.
* Incremented when NAK is received in response to a block transmission.

RXGOODS:
* Initialised to 0 when an ENQ to establish a data link is received.
* Incremented when the received CRC checksum validates against the computed CRC checksum.

RXBADS:
* Initialised to 0 when an ENQ to establish a data link is received.
* Incremented when the received CRC checksum does not validate against the computed CRC checksum.

6.7.8 Configuring the Protocol (SETSYNC8)

For maximum flexibility there are a number of features which may be controlled by the user These are controlled by a number of flags which are described in detail later in this section. These flags are stored in RAM (as detailed in Part 4) and are preserved just like the other communications parameters such as transmission speed.

They may be changed by a user program writing direct to memory (eg POKE in BASIC). Alternatively, the BASIC program "SETSYNC8" can be used to configure the protocol using a menu of options. SETSYNC8 is listed later in this section.

Using SETSYNC8

SETSYNC8 is a **Hunter** BASIC program and should be run accordingly.

NOTE: On versions earlier than 9G08, use SETSYNC, listed in Appendix X.

The options offered by SETSYNC8 are as follows:

* RECORD LENGTH - Default 80: The record can be set in the range of 1-255. The record length has to be at least 9 less than the buffer length to allow for the protocol control characters. This option can be varied for host machine compatibility.

* BUFFER LENGTH - Default 200: The buffer length can be set in the range of 10-512. The buffer length can be altered to accommodate for good or bad transmission conditions.

* SINGLE RECORD FLAG - Default NO: If this option is set to NO, multiple records can be sent in a block. Alternatively, with the YES option, a single record is sent in each block.

* RECORD PADDING - Default NO: This option, if set to YES, will pad out a record with spaces to its maximum length, (as selected by option 1), before transmission and strip off trailing spaces on reception. If set to NO, record padding is inhibited and trailing spaces are not stripped.

* US OR IRS - Default US (2780): These options determine the essential difference between the 2780 and 3780 protocols. The selection is between terminating a record with US or with IRS (both followed by CRs) for 2780 and 3780 respectively

* INSERT STX - Default NO: If the option is set to YES, a STX character is inserted at the start of every intermediate record. If NO, a STX character is only inserted at the start of the first record in a block. Intermediate STX's are ignored on reception.

* RETRYS - Default 10: This option can be set to either 10 or infinity. When set to 10 and a communications error is detected, the **Hunter 2** will retry sending the last block ten times. If set to infinity, the **Hunter 2** will retry indefinitely and will continue until it succeeds or is switched off.

* SPACE COMPRESSION - Default NO: This option can be set to YES to compress a string of more than two spaces into an IGS character and a space count code on transmission and the reverse on reception.

* DEFAULT VALUES: Selecting this option sets the parameters to the 2780 default values, as described above.

SETSYNC8 Listing

```
5 IFPEEK(57276)>1THENPOKE57276,1
10 KEYOFF:CLS:CUROFF:ONBREAKGOTO90
20 OPCHR15,15,0:PRINT"Bisync  Parameters"
30 PRINT"1 Record Len...      6 Int STX......."
40 PRINT"2 Buffer Len...      7 Retrys........"
50 PRINT"3 Single Rec...      8 Spc Compress.."
60 PRINT"4 Rec Padding..      9 Ac Coupler...."
70 PRINT"5 EM or IRS....      0 TRANSPARENCY.."
80 PRINT"e Exit........."
82 PRINT"e Exit........."

85 GOSUB900
90 GOSUB800:CURON:INCHR"  Your Choice ";YC:CUROFF
99 IFYC<>101THEN100ELSEA=ARG(38):A=CALL(5):A=CALL(0)
100 IFYC<48ORYC>57THENBEEP:BEEP:GOTO90
110 YC=YC-48:ONYCGOTO200,250,300,350,400,450,500,550,600
120 A=PEEK(57276):A=AXOR1:POKE57276,A
130 OPCHR15,37,5:IFATHENPRINT"ON "ELSEPRINT"OFF"
140 GOTO90

200 REM SET RECORD LENGTH
210 GOSUB800:CURON:INPUT;"Enter Record Length ";RL:CUROFF
220 IFRL<10ORRL>255THENGOSUB800:BEEP:PRINT"LENGTH OUT OF RANGE";:GOTO248
230 IFRL>=BLTHENGOSUB800:BEEP:PRINT"LENGTH GREATER THAN BLOCK LENGTH";:GOTO248
240 POKE63571,RL:OPCHR15,14,1:PRINT"    ";:OPCHR15,14,1:PRINTSTR$(RL);" ";
245 GOTO90

248 FORI=0TO600:NEXTI:GOTO210
250 REM SET BUFFER LENGTH
260 GOSUB800:CURON:INPUT;"Enter Buffer Length ";BL:CUROFF
270 IFBL<100RBL>512THENGOSUB800:BEEP:PRINT"LENGTH OUT OF RANGE";:GOTO298
280 IFBL<=RLTHENGOSUB800:BEEP:PRINT"LENGTH LESS THAN RECORD LENGTH";:GOTO298
290 POKE63572,BL-256*FIX(BL/256),FIX(BL/256)
295 OPCHR15,14,2:PRINT"    ";:OPCHR15,14,2:PRINTSTR$(BL);" ";:GOTO90
298 FORI=0TO600:NEXTI:GOTO260

300 REM SELECT SINGLE RECORD BUFFER
310 IFPEEK(63574)=0THENPOKE63574,255ELSEPOKE63574,0
320 OPCHR15,14,3:IFPEEK(63574)=0THENPRINT"NO "ELSEPRINT"YES"
330 GOTO90

350 REM SELECT RECORD PADDING
360 IFPEEK(63575)=0THENPOKE63575,255ELSEPOKE63575,0
370 OPCHR15,14,4:IFPEEK(63575)=0THENPRINT"NO "ELSEPRINT"YES"
380 GOTO90

400 REM SELECT EM OR IRS
410 IFPEEK(63576)=0THENPOKE63576,255ELSEPOKE63576,0
420 OPCHR15,14,5:IFPEEK(63576)=0THENPRINT"EM "ELSEPRINT"IRS"
430 GOTO90

450 REM SELECT INTERMEDIATE STX
460 IFPEEK(63577)=0THENPOKE63577,255ELSEPOKE63577,0
470 OPCHR15,37,1:IFPEEK(63577)=0THENPRINT"NO "ELSEPRINT"YES"
480 GOTO90

500 REM SELECT RETRYS
510 IFPEEK(63578)=0THENPOKE63578,255ELSEPOKE63578,0
520 OPCHR15,37,2:IFPEEK(63578)=0THENPRINT"10 "ELSEPRINT"INF"
530 GOTO90
```

```
550 REM SELECT SPACE COMPRESSION
560 IFPEEK(63579)=0THENPOKE63579,255ELSEPOKE63579,0
570 OPCHR15,37,3:IFPEEK(63579)=0THENPRINT"NO "ELSEPRINT"YES"
580 GOTO90

600 REM SELECT ACOUSTIC COUPLER
610 IFPEEK(63580)=0THENPOKE63580,255ELSEPOKE63580,0
620 OPCHR15,37,4:IFPEEK(63580)=0THENPRINT"NO "ELSEPRINT"YES"
630 GOTO90

800 REM WIPE OUT BOTTOM LINE OF SCREEN
810 OPCHR15,0,7:PRINTSPACE$(39);:OPCHR15,0,7:RETURN

900 REM INITIALIZE SCREEN VARIABLES
910 RL=PEEK(63571):BL=PEEK(63572)+256*PEEK(63573)
914 OPCHR15,14,1:PRINTSTR$(RL);
916 OPCHR15,14,2:PRINTSTR$(BL);
920 OPCHR15,14,3:IFPEEK(63574)=0THENPRINT"NO "ELSEPRINT"YES"
930 OPCHR15,14,4:IFPEEK(63575)=0THENPRINT"NO "ELSEPRINT"YES"
940 OPCHR15,14,5:IFPEEK(63576)=0THENPRINT"EM "ELSEPRINT"IRS"
950 OPCHR15,37,1:IFPEEK(63577)=0THENPRINT"NO "ELSEPRINT"YES"
960 OPCHR15,37,2:IFPEEK(63578)=0THENPRINT"10 "ELSEPRINT"INF"
970 OPCHR15,37,3:IFPEEK(63579)=0THENPRINT"NO "ELSEPRINT"YES"
980 OPCHR15,37,4:IFPEEK(63580)=0THENPRINT"NO "ELSEPRINT"YES"
985 OPCHR15,37,5:IFPEEK(57276)THENPRINT"ON "ELSEPRINT"OFF"
990 RETURN
```

Record Length (RECLEN)

This is a single byte which defines the maximum record length. It has a range of 1-255, with a default of 80. When outputting data, this length should not be exceeded as there may be insufficient space left in the block to accommodate the record.

If record padding is used, all records are either padded or truncated to fit this length.

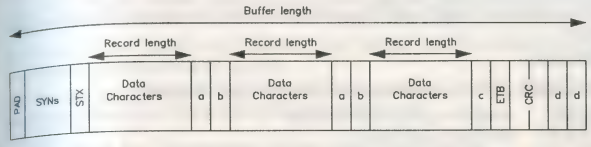
Buffer Length (BUFLEN)

This is a two byte number which defines the maximum length of a block. It has a permitted range of 10-512 with a default of 200.

The block length must be greater than the selected record length. This control is useful for adjusting for telephone line quality, when transmitting over modems or acoustic couplers. For a good line, the buffer length could be set to maximum so that line turnarounds are kept to a minimum and the maximum data is sent in one block. For a bad line, the buffer length is set to a single record length, to minimise the amount of data to be re-transmitted in the event of a NAK or data corruption due to the bad line.

On receive, the block length can be anything up to 512 characters.

Buffer Organisation



a Record terminator, depends on the state of the mode flag: either an IRS character or an IUS character followed by 2 intermediate characters

b If intermediate STX is selected, an STX character.

(Note that this will not be counted as part of the record length.)

c An IRS or EM, if selected.

d Line turnaround characters (FFH).

When calculating the buffer length, space should be included for the characters at the start of the buffer (ie PAD, SYNs and STX), the characters at the end (ie ETB or ETX, 2 CRC characters and 2 line turnaround characters) and all the required record terminators and intermediate STXs.

Single Record Flag (SRFLG)

This flag controls whether there is single or multiple records in a block. If set to zero (default value) there will be multiple records in each block. If set non-zero, there is only one record transmitted per block.

Record Padding Flag (RPFLG)

This single byte flag is used to control record padding. If this byte is set to non-zero, a record is padded with spaces to its maximum length as defined by record length before being transmitted. On receiving a record, trailing spaces are stripped off. If this flag is set to zero, record padding is inhibited and trailing spaces are not stripped on reception. The default state of this flag is zero.

Record padding is only normally used on fixed length 2780 communications.

Mode Flag (EMFLG)

This flag determines the essential difference between 2780 and 3780. It selects between terminating a record with US (1FH), followed by CRC, or with IRS (1EH) followed by CRC.

2780 mode is the default with the mode flag set to 0. A non-zero flag selects 3780 mode.

2780 mode:

Fixed length records are terminated by any of the following:

US,CRC	used for a partly filled buffer
ETB,CRC	used for a full buffer
ETX,CRC	used for the last buffer to be sent

Variable length records are terminated by any of the following:

EM,US,CRC	used for a partly filled buffer
EM,ETB,CRC	used for a full buffer
EM,ETX,CRC	used for the last buffer to be sent

3780 mode:

All records are terminated with IRS.

On receive, record terminators (CR) are always returned for US CRC or IRS, irrespective of the setting of the mode flag.

STX Flag (STXFLG)

This flag is a single byte and controls the use of intermediate STXs. If the flag is set to non-zero, at the start of every intermediate record an STX is inserted. If the flag is zero, then STXs are not inserted except at the start of the first record in a block. Intermediate STXs are ignored on reception. The default setting of this flag is zero.

This feature is normally only used in 2780 installations.

Retry Flag (RETRYS)

This single byte flag controls how the protocol handles errors.

If RETRYS is set to zero (its default state), **Hunter 2** will send a block again up to 10 times before giving a communication error.

If RETRYS is set to non-zero, the **Hunter 2** will re-try indefinitely until either it succeeds or the operator switches it off.

Space Compression Flag (SCFLG)

This single byte flag controls the operation of space compression.

If the flag is non-zero, any string of more than 2 spaces is compressed into an IGS character and a space count code on transmission. The reverse happens on receive (ie the codes are expanded to the string of spaces).

If the flag is set to zero (its default state) spaces are not compressed on transmission or expanded on receive.

This feature is not normally used in 2780 installations. It can be used to give increased throughput when data contains many spaces.

6.7.9 Use In Terminal Emulation Mode

Although 2780 workstations are generally not used interactively, you may use **Hunter 2** interactively, for example, to test the system with LOG ON cards, etc.

In terminal emulation mode, any received data is displayed on the screen. To transmit a message, type it directly. Provided no mistakes are made, and there is less than 5 seconds between each character, the message is sent on the transmission channel. (If you wait more than five seconds, the incomplete message will be sent.)

6.7.10 Hardware Handshaking

Hardware handshaking is generally used with modems or modem simulators. The use of RTS (Request to Send) and CTS (Clear to Send) is therefore determined by the modem, particularly if half duplex or two wire operation is desired (as on the public switched telephone network). Both RTS and CTS may be deactivated as usual.

If they are used, then the following takes place:

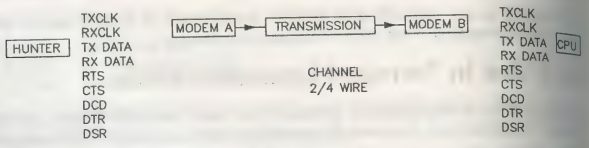
1) If **Hunter 2** is not transmitting, RTS is inactive, allowing the remote modem to transmit.
2) When starting a transmission buffer, RTS is activated. Before transmission starts, **Hunter 2** waits for CTS from the modem. When CTS becomes active **Hunter 2** starts transmission.
3) During transmission RTS is kept active, but CTS is not checked (as if transmission stopped when it became inactive, synchronism would be lost).

4) At the end of a transmission, RTS is deactivated and **Hunter 2** waits for reception.

There are no timeouts on handshaking.

6.7.11 Hardware Configuration

The transmission channel could be the switched public network, 4 wire private lines, etc.



The two modems and the channel could be a modem eliminator for nearby operation.

As can be seen from the above diagram, the modems generally create all the clocking in this simple arrangement. **Hunter 2** will not generate any actual clocking of its own, except when self synchronising to the data.

The handshake signals RTS and CTS ensure correct data flow over a half duplex line (allowing the modems to settle etc.) They may not be necessary over 4 wire links.

6.7.12 Hunter 2 to Hunter 2 Connection

To operate this protocol back-to-back between two **Hunter 2**s you need only use a crossed 3 wire lead (TX, RX and ground). If the handshake lines are connected, deselect CTS on the menu, as the handshaking is only configured for modem operation.

The two **Hunter 2**s must be set for self-clocking at the designed speed, typically 1200 baud or 300 baud over slow speed asynchronous modems.

6.8 Communications Setup Error Messages

The configuration menu does not display any error messages as such, as it only allows you to select legal options.

If **Hunter 2** beeps while using the menu, you have pressed an unacceptable key. Use the cursor keys and [ENTER].

The error messages which may be displayed while using communications are described in Chapter 7.

Chapter 7 Using Communications

7.1 Introduction

Hunter 2's communications port can be used to connect to a very wide range of devices. These fall into two main categories: other computers and computer peripherals.

This chapter describes how to use the port once you have set it up and connected to the other device, as described in Chapter 6. Refer to Chapter 6 for background information on communications, too.

7.1.1 Using Communications

Hunter 2's communications can be used for exchanging programs and data with other systems. For example:

Programs can be loaded into **Hunter 2** daily, or as required, from a central database. Secure protocols and program-based error checks can guarantee accurate copies.

New, revised or corrected programs can be returned to the database for storage. Database files and command functions can be accessed automatically, or under manual control using **Hunter 2**'s terminal emulation.

Data can be collected and down-loaded to mainframe, minicomputer or microprocessor storage whenever required. Data transmissions can be formatted exactly to resemble existing data terminal configurations, allowing **Hunter 2** to slot directly into existing software support structures.

Database information (for instance, tables of names and addresses) can be sent to **Hunter 2** for presentation to operatives in the field. Collected data can be linked to previously loaded database information for instant verification and validation.

7.1.2 Standard applications

A number of communications applications are supplied as part of DEMOS:

SEND transmits a file, for example to store it on another computer or print it.

INP inputs a file from another computer

TERM makes **Hunter 2** behave as a portable computer terminal, to allow you to use a minicomputer or mainframe.

CRT allows you to operate **Hunter 2** from a computer terminal.

The port may also be used from standard CP/M programs, as logical device LST· or PUN: (output) and logical device RDR: (input). For example, most applications will send printed output to the port.

7.1.3 Programming

User programs can use communications almost as easily as console input/output, allowing the communications package to handle the complexities of data rates, handshaking, protocols and error correction.

The execution of user programs is largely unaffected by communications activity, because the communications package is driven automatically by interrupts generated by the port, as and when data is received or may be transmitted. In normal operation, the only effect that you might see is a slight slowing down of program operation, if a lot of data is being transferred.

The details of using communications from BASIC programs are described in Part 3. The details of using communications from assembly language or compiled programs are described in Part 4.

7.1.4 Starting communication

When you turn **Hunter 2** on, the serial output interface is left turned off to conserve battery life. **Hunter 2** turns it on automatically when a user program attempts to send or receive data, or if the handshake lines become active. **Hunter 2** waits approximately 2 seconds to allow the interface to settle down before beginning transmission or reception.

While the port is turning on it may send spurious characters to the remote system, which may cause problems. To avoid this you may turn the port on before connecting to the remote system. For example, by executing a BASIC LPRINT command.

The port remains turned on until you turn the **Hunter 2** off, so further transmissions are not affected in this way. (To save power you may also turn off the port under program control, as described in Parts 3 and 4.)

7.2 Transferring Files

The most important facility of a portable computer is its ability to communicate with other computers and peripheral devices. **Hunter 2** uses the industry-standard RS232 interface and supports the most common and some of the not so common protocols and baud rates. This allows it to communicate with a wide range of computers, including microcomputer systems such as the IBM-PC.

Hunter 2 can load files using two different methods: 8-bit (binary) transfer and Intel Hex format transfer.

8-bit transfer involves sending the files direct from the other computer The computer's port and the utility used must support 8-bit transfer For example, if the PIP utility is used it must have the [O] option to indicate that all 8 bits must be sent.

Intel Hex format transfer is two and a half times slower than 8-bit transfer but has the advantage of built-in checksums to guarantee the accuracy of data. To use it, you must first convert the files to Intel Hex format (described in Appendix VII) on the other computer. You then transfer these files to **Hunter 2** in the normal manner, which converts them back to the original binary format for you to use.

These two methods are described in full in sections 7.2.1 to 7.2.3.

NOTE: Some CP/M systems may not be capable of 8-bit communication. Check the serial port hardware manual for details of the UART configuration if in doubt. If 8-bit communication is not available, use Intel Hex format to transfer binary files.

7.2.1 File Description

In the examples, it is assumed that the application to be loaded into the **Hunter 2** consists of the following three files:

Filename	Type	Length(k)	Blocks	Records
MAIN.COM	Object	22	88	176
OVERLAY.OVR	Object	10	40	80
PRODUCTS.TXT	ASCII	14	56	112

The program MAIN.COM is a file which organises and sorts a database file PRODUCTS.TXT The file OVERLAY.OVR is an overlay file which like MAIN.COM is an executable program, but parts are loaded into the TPA as and when required by MAIN.COM. All three files must be present in the filespace for correct operation.

7.2.2 8-bit Transfer

Step 1

Set up the communications as follows (see 6.1).

Baud rate	As required.
Protocols	None.
Parity	8-bit.
RTS	As required.
SERIG	Off.

Step 2

Determine the size of the files to be copied.

```
STAT *.* [ENTER]

Recs   Bytes   Ext   Acc
176    22k     2     R/W    A:MAIN.COM
80     10k     1     R/W    A:OVERLAY.OVR
112    14k     1     R/W    A:PRODUCTS.TXT
```

Hunter 2 loads files in blocks which are twice the size of Records. Therefore MAIN.COM is 88 blocks in length.

Step 3

Load MAIN.COM.

Connect **Hunter 2** to the host computer with a suitable (5 way crossed) RS232 cable. Then, on **Hunter 2**, type:

```
INP 88 MAIN.COM [ENTER]
```

Hunter 2 will display·

```
* Waiting
```

To abort use the ESC key. Assuming that PIP is used to transfer the file to **Hunter 2** via its LST: printer port, type on the host computer:

```
PIP LST:=MAIN.COM[O] ENTER
```

The [O] suffix is very important because this tells PIP to output using the full 8 bits. **Hunter 2** will respond to the beginning of transmission with the message:

```
* Loading
```

If you specified too many blocks in the INP command, **Hunter 2** will display a `* Waiting` message when transmission has finished. Press the ESC key to complete the transference of the file.

You have now loaded the file MAIN.COM onto **Hunter 2**'s disk.

Step 4

Now load OVERLAY.OVR.

On the **Hunter 2**, type:

```
INP 40 OVERLAY.OVR ENTER
```

On the Host, type:

```
PIP LST:=OVERLAY.OVR[O] ENTER
```

Step 5

Now load PRODUCTS.TXT.

On the **Hunter 2**, type:

```
INP 56 PRODUCTS.TXT ENTER
```

On the Host, type:

```
PIP LST:=PRODUCTS.TXT[O] ENTER
```

Step 6

Finally, to check that the application has transferred correctly, run the program on the **Hunter 2** ie type:

```
MAIN ENTER
```


It is important to check the transferred files, because 8-bit transfer does not include any error-checking.

7.2.3 Intel Hex Format Transfer

Step 1

You must convert all the files to Intel Hex data format before transmitting them. This is achieved by using a utility supplied with most CP/M systems (usually called UNLOAD). If you do not have access to UNLOAD, a version is reproduced in Appendix VI.

Once you have converted the files you will have two versions of each. The versions in Intel Hex format will usually have the extension .HEX.

MAIN.COM	MAIN.HEX
OVERLAY.OVR	OVERLAY.HEX
PRODUCTS.TXT	PRODUCTS.HEX

Step 2

Set up the communications as follows (see 6.1):

Baud rate	As required
Protocol	NONE
Parity	As required
RTS	As required
SERIG	OFF

Step 3

Load MAIN.COM.

Connect **Hunter 2** to the host computer with a suitable (5 way crossed) RS232 cable. Then on **Hunter 2** type:

```
INP ENTER
```

Hunter 2 will display:

```
* Waiting
```

To abort use the ESC key. Assuming that you are using PIP to transfer the file, via the computer's LST: printer port, type the following on the host computer:

```
PIP LST:=MAIN.HEX
```
(ENTER)

As soon as transmission begins, **Hunter 2** displays:

```
* Loading
```

The file is not loaded directly into a **Hunter 2** file, but into the TPA. When the file is loaded, **Hunter 2** displays:

```
* Loading completed
File length = 88 Blocks
```

You can now save the data into a file with the SAVE command. In this case:

```
SAVE 88 MAIN.COM
```
(ENTER)

You have now created the file MAIN.COM.

Step 4

Load OVERLAY.OVR.

On the **Hunter 2**, type:

```
INP
```
(ENTER)

On the Host, type:

```
PIP LST:=OVERLAY.HEX
```
(ENTER)

When transmission has finished, on the **Hunter 2**, type:

```
SAVE 40 OVERLAY.OVR
```
(ENTER)

Step 5

Load PRODUCTS.TXT:

On the **Hunter 2**, type:

```
INP
```
(ENTER)

On the Host, type:

```
PIP LST:=PRODUCTS.HEX
```
(ENTER)

When transmission has finished, on the **Hunter 2**, type:

```
SAVE 56 PRODUCTS.TXT
```
(ENTER)

Step 6

Finally, to check that the application has transferred correctly, you can run the program on the **Hunter 2** ie type:

```
MAIN
```
(ENTER)

This should not be necessary, because Intel Hex transfer incorporates automatic error-checking. If an error is detected during transfer, **Hunter 2** will display the message:

```
* Loading error
```

(If this happens, transfer the file again.)

7.3 Using Hunter 2 as a Terminal

7.3.1 Introduction

In terminal emulation mode, **Hunter 2** can be used as:

* A remote terminal for dialup timeshare computer services.
* As a peripheral to other computers.
* As a portable 'Telex', able to communicate with other **Hunter 2**s or data terminals.

7.3.2 Connecting to the Host

Hunter 2's RS232 port is configured to represent a Data Terminal Equipment (DTE) for compatibility with modems and other items. For further details, see Part 5.

Before using terminal emulation, check that the communication port is configured correctly (see section 6.3).

7.3.3 Selecting Terminal Emulation

Terminal emulation is normally selected from the File Manager, by typing:

TERM [ENTER]
or
[CTL/FN] 6

Hunter 2 is now in terminal emulation mode, ready to communicate with the host computer

(Terminal emulation may also be selected by a user program: see Part 4 for details.)

In terminal emulation mode, the function keys are programmed as follows:

[CTL/FN] 4 (COMS) leaves terminal emulation temporarily to allow you to reconfigure the port.

[CTL/FN] 8 (SYST) leaves terminal emulation and returns to the File Manager (or other calling program).

Any other keys cause characters to be transmitted, including control characters. Their effects depend very much on the host and the protocol being used. Upper and lower case alphabets are available.

Characters sent by the host will be displayed if printable. The host will usually send back (echo) any text you type. If it does not, you can achieve a similar effect by configuring transmission with 'ECHO:y'

To leave terminal emulation mode, use the SYST function key

7.3.4 Protocols

Any protocols selected will be observed automatically by **Hunter 2**'s communication software package. Remember that block protocols like ACK/NAK will not transmit until you press [ENTER].

7.3.5 Buffering

It should be remembered that the **Hunter 2** is fully buffered with transmit and receive communications buffers, a keyboard buffer and a virtual screen effectively keeping early information available for use.

At high receive data rates, eg 4800 baud, the receive buffer will remain full. If a "stop" is sent to the sending computer (eg XOFF or control S, sent manually) then the screen display will not stop immediately, continuing to display the characters stored in the receive buffer until the buffer empties. If this is found undesirable, the most straightforward solution is to reduce the data rate to 1200 baud.

If 2780 synchronous transmission is used in terminal mode, a block will be sent if the keyboard has not been operated for 5 seconds. This does mean that when using terminal emulation in 2780 mode for "signing-on", there should be no hesitation or typing errors or the block will be sent unexpectedly!

7.3.6 Auto Power Off

When using terminal emulation, **Hunter 2** will *not* power off automatically. This is for ease-of-use when operating intermittently with a remote site. If auto power off were allowed, the modems might drop the line, resulting in loss of messages, etc.

It is important, therefore, to remember to switch off **Hunter 2** after using Terminal Emulation, to preserve battery power

7.4 Advanced Communications Techniques

7.4.1 Introduction

This section describes certain aspects of using communications that require a greater understanding of the mechanisms involved and simple programming. These techniques will generally be used by applications writers.

7.4.2 9600 Baud Communications

Although **Hunter 2** is designed to communicate at a maximum rate of 4800 baud, it can also be used at 9600 baud, in certain applications, as follows.

Communication at 9600 baud must be half duplex and be controlled by hardware handshaking; asynchronous protocols may not be used.

It is essential that the communication is ONE way only; communication in duplex mode will cause data corruption, as the interrupts sampling the data will be offset from the ideal point to sample the data.

On power up, the operating system defaults the communications parameters to those selected in the communication menu. It is, therefore, necessary to place the routine which sets communications to 9600 baud half duplex so that it is executed each time the **Hunter 2** is powered up.

It should also be noted that CP/M calls such as call 38 to set up the RS232 port, and call 41 to initialise communications will also cause the setting of the communications to the default values. Additionally DEMOS functions such as INP and the EDITOR will re-initialise the communications parameters on entry.

Transmission

To select 9600 baud transmission, program the communications timer by executing:

```
OUT 16,161
OUT 17,1
```

Reception

To select 9600 baud reception communications, program the timer registers by executing:


```
POKE ad,0,0,0,0
X=161
POKE 63591,X,1:
POKE 63586,1
```

For versions 9G and 9G06, ad is 57740. For version 9G08, ad is 57676.

The tolerance on reception is very small. If characters are corrupted, vary the parameter X in the range 125-161 to optimise reception.

WARNING

9600 baud communications will only work in one direction at a time. Attempting communication in two directions simultaneously will usually result in the operating system turning the **Hunter 2** off, because insufficient time is available to handle the interrupts.

If applications require software handshaking, they must ensure that when the handshake character is transmitted the other communications device is not also transmitting. This could be achieved by the host delaying several milliseconds after it has received the last character, before sending a reply. A similar turnaround delay should also be utilised if the **Hunter 2** has received a buffer, before it transmits its reply character.

If transmitting data, reception of data could be inhibited by disabling the receiving interrupts prior to data being sent. This will ensure that no data sent to the **Hunter 2** can cause problems. To do so, execute:

```
POKE 63888,8: REM set INTMSK location
OUT 187,8:    REM enable A interrupt only
```

To reinstate the interrupts correctly, use CP/M call 41.

7.4.3 Using Bisynchronous Protocols

Transparent Data and Internal Clocking (9G08 only)

When receiving transparent data blocks, **Hunter 2** may receive long sequences of 0FFH characters or 00H characters. This may cause problems if using internal clocking, as such data does not contain any bit transitions for **Hunter 2** to synchronise to, so it may lose synchronism and reject a block.

NOTE: Even if the **Hunter 2** does lose synchronism and reject a block, the protocol will ensure that no faulty data will be accepted.

If such data may be sent, to avoid this potential problem, restrict the maximum block length to 300 characters, as follows.

2780 Block Format

If EMFLAG is set to zero, **Hunter 2** will use 2780 block format. This format terminates each intermediate record in a block with DLE IUS and an intermediate CRC character sequence. As all records are terminated by this sequence and the maximum record length supported by the **Hunter 2** is 255 characters, no problems should be encountered in using transparency in 2780 mode.

However, Husky Computers recommend that you set the record length to 80 characters to maintain compatibility with other 2780 type equipment. The block length is not critical, as the DLE IUS sequence at the end of each record contains bit transitions that the **Hunter 2** can re-synchronise on. The following **Hunter** BASIC program will configure the protocol as discussed above.

```
10 POKE 63576,0: REM Select 2780 mode
20 POKE 63571,80: REM Set record length to 80
30 POKE 63572,200,0: REM Set block length to 200
40 POKE 57276,255: REM Select transparency
50 A=ARG(38):A=CALL(5): REM Activate transparency
```

3780 Block Format

If EMFLG is set to non-zero, **Hunter 2** will use 3780 block format. This format specifies that no terminator is used after each intermediate record. Thus it is possible to have a continuous stream of data characters which is more than 300 characters long.

To avoid loss of synchronism, Husky Computers recommend that a record length of 80 characters is used and that the single record flag is set non-zero. This will ensure that only one 80 character record will be sent in a block. The block length should be set to some suitable figure over 90 characters (eg 100) to ensure that

the maximum number of characters that could be received without bit transitions is 80 characters. The following **Hunter** BASIC program will configure the protocol as discussed above.

```
10 POKE 63576,255: REM Select 3780 mode
20 POKE 63571,80: REM Set record length to 80
30 POKE 63572,100,0: REM Set block length to 100
40 POKE 63574,255: REM Select single records
50 POKE 57276,255: REM Select transparency
60 A=ARG(38): A=CALL(5) : REM Activate transparency
```

It should be noted that exceeding that above limit could cause the **Hunter 2** to loose synchronism with incoming data, resulting in the **Hunter 2** rejecting the block as corrupted. The **Hunter 2** will not accept a faulty block as correct, due to the protocol's CRC checking.

Noisy Lines (9G06 only)

NOTE: This problem does not occur with version 9G08 or later.

The bisynchronous communication protocol defines two block received acknowledgements, ACK0 and ACK1, which are used alternately to acknowledge blocks, and NAK, which is used to request re-transmission.

If a version 9G06 **Hunter 2** is transmitting over a noisy communications line, it may interpret a corrupted ACK as a NAK and transmit the block again. However, the other system does not know this is a duplicate block (as it did not request one), so acknowledges with the next acknowledgement character

Hunter 2 will refuse to accept this acknowledgement, as it is the wrong type. A dialogue of ENQ-ACK then occurs until the Husky finally declares a communications failure.

The frequency of this fault depends on the quality of the PSTN line. If the receiving station is sophisticated enough, the total communications failure can be minimised by detecting the occurrence of the ENQ-ACK dialogue, and modifying the ACK response after rejecting the misinterpreted block.

Alternatively, the **Hunter 2** program can utilise the ON COMMS trap to detect a communications failure and reinitialise communications once more and re-transmitting the data.

Controlling Unanswered Line Bids In Applications (9G08 only)

When using the IBM3780 protocol, if no reply is received to the ENQ bid at the start of transmission, an 'unanswered line bid error' occurs. This cannot be trapped by application programs in the way that other communication errors can.

When this error occurs, the prompt is displayed:

```
Unanswered line bid
Retry transmission (Y/N) -
```

This allows an operator to indicate whether to retry or to abort the communications. This may however be inconvenient, particularly in applications which are designed to run unsupervised eg overnight.

This section details how to trap an unanswered line bid and take remedial action, without operator intervention.

Principles

When IBM3780 commences communications, it issues the line bid ENQ at approximately five second intervals, looking for an acknowledgement reply ACK0 from the other device.

Providing the ENQ requests are not set to infinite retries, when communications are started the ENQ counter location ENQCNT is set to zero. When the transmission occurs the ENQCNT location is set to the value 10, and there after it is decremented for each ENQ sent at approximately five second intervals, providing no acknowledgement is received.

When the location ENQCNT reaches zero the communications handler goes immediately to the unanswered line bid screen requesting an input from the user.

In order for an application to trap that the communications handler is failing to establish a communications link, the application can start communications and then monitor the progress of the ENQCNT location.

When the ENQCNT location reaches a value of one, the application could decide that the communications handler is not going to establish the link and goto an error handling routine.

It is important that the first action of the error handling routine is to re-initialise communications by using CP/M function 38, in order to abort the line bid and reset communications to the quiescent state.

Obviously the loop monitoring ENQCNT will also need to detect if an acknowledgement has been received to the line bid. This can be achieved by monitoring also the location TXGOODS, which will be incremented when an acknowledgement is received. This check can be simplified by setting TXGOODS to zero prior to starting communications, and then monitoring for a non-zero change of the location indicating that communications have commenced successfully

The locations mentioned are as follows:

```
ENQCNT  EQU FAB3H (1 byte)
TXGOODS EQU DFB4H (2 byte)
```

Example BASIC Demonstration Program

The following BASIC program demonstrates the principles explained above. It assumes that the communications parameters are already set for IBM3780.

```
10  REM clear TXGOODS location
20  POKE 57258,0 : POKE 57259,0
30  REM start communications
40  LPRINT "hello"
50  IF PEEK (64179) = 1 THEN GOTO 1000
60  IF PEEK (57268) <> 0 THEN GOTO 100
70  GOT 50 : REM loop awaiting comms
100 REM communications started OK
110 REM here will go main body of code
120 END
1000 REM unanswered line bid error handler
1010 A = ARG (38) : A = CALL (5) : REM reset comms
1020 CLS
1030 PRINT "*** unanswered line bid ***"
1040 REM here will go error handler in application
1050 END
```

Applications In Other Languages

The principles explained above and demonstrated by the example BASIC program could also be implemented in machine code or compiled languages.

It is important to ensure in such applications that while waiting in the loop, a call to keyboard status (CP/M function 6 or CP/M function 11) is executed. This ensures that DEMOS maintains the software timers involved in the communications parameters, which reside in the keyboard scan routines.

An Alternative Method Using The Keyboard Buffer

An alternative method can also be used, in machine code or compiled code applications only This forces the character 'n' into the keyboard buffer before beginning communications, so that if an unanswered line bid error occurs, the negative response 'n' will be read as if typed by the operator, causing the error to be handled by the communications error 1 routine, which can then be trapped by the application.

This method has the advantage that it can be used at any time during the communications, and so trap any unanswered line bid during transmission.

Take care during communications not to perform any keyboard input calls which will remove the character from the buffer

This method **cannot** be used with **Hunter** BASIC, as the interpreter automatically removes characters from the keyboard buffer during program execution.

If entering a loop during communications it may be necessary to perform a keyboard read in order to maintain the timer routines which are located at the front of the keyboard scanning routines.

The following is a simple machine code demonstration program of how to force the buffer

```
        kbdon  : equ 0dfb0h
        kbdoff : equ 0dfb2h
        kbdbuf : equ 0df80h

start:  ld   c,4
        ld   e,'a'
        call 5h                 ;start comms

loop:   ld   hl,kbdbuf          ;force keyboard
        ld   (hl),'n'
        ld   (kbdoff),hl
        inc  hl
        ld   (kbdon),hl
        ld   e,-1               ;maintain timers
        ld   c,6
        call 5h
        jp   loop
```

7.4.4 Using the SYSTIME Protocol

While IBM bisynchronous 2780/3780 protocol formats are strongly recommended for high-performance telephone communications, **Hunter 2** also supports a number of lower performance asynchronous protocols. These are often more convenient where the host computer does not support bisynchronous protocols, allowing protocol handlers to be implemented easily on the host computer, even in languages like BASIC.

One protocol built into most Husky models as a standard feature is "SYSTIME". However, this protocol has a couple of deficiencies which make it less reliable than ideal for practical PSTN communication systems. These are as follows:

* There is no block counting function, so if an acknowledge is missed, the system may re-transmit the same block. As the receiving system cannot recognise that the second block is a duplicate, it will interpret it as a normal block, leading to duplication of data and mis-sequencing. This is due to the system not having either the two alternate acknowledges of the bisynchronous protocol or a built-in block counter of some other asynchronous systems.

* In a two-way communication system, there may be problems when **Hunter 2** finishes transmission and waits reception. This is due to there only being one acknowledgement for the last transmitted block. If this is missed by the transmitting machine, then it will send the block again. This can mean that it will never receive an acknowledgement as the receiving system has finished communicating and may have shut down the communications line after it has transmitted the final acknowledgement. This will eventually lead to a communications error on the transmitting machine, when there is in fact none.

Solutions to these problems can be effected from within the user program. To solve the first problem, all that is required is for the user program to organise transmitted data into 64 byte records and append an incrementing record number to each record. This can then be checked by the receiving system to check for correct blocks. Hence, to transmit - it is recommended to use a 64 byte string and utilise the first byte of that string as a counter from, say, 0-9. The receiving system will check this first character of each 64 byte string and establish whether it had already received the block or not. If it has, then it merely ignores the duplicate block and waits for the new one. This provides a very simple solution.

There is no clear-cut answer to problem 2 in any protocol, as far as can be established. This is because the system has to stop transmitting sometime, or stop receiving sometime and something has to be the last thing to transmit to shut down the line. The bisynchronous protocol overcomes this problem by using a specific ETX character, but even this is not completely error free. To alleviate the difficulty, the receiving machine should send out a stream of acknowledges when it detects the last block. This will undoubtedly get through to the transmitting machine.

To achieve this, switch protocols from SYSTIME to NONE and transmit a stream of ACKs (06H).

Implementation of these two techniques will ensure that the SYSTIME protocol can be used as a practical communication medium for reliable telephone network communications.

7.5 Communications Errors

Hunter 2 provides a range of communications error messages to warn the user of failure situations. The error messages are designed to be trapped by the BASIC ONCOMMS statement (see Part 3) and handled by the user's program.

However, errors can also occur when **Hunter 2** is under manual control (for instance, when used in terminal emulation mode) or the user program does not contain ONCOMMS. In this event, **Hunter 2** will display:

```
* * * Communication Failure * * *
        - - Error nn - -
```

where *nn* is the Error number, in hexadecimal. To cancel the attempt at communication turn off the **Hunter 2** or press [ENTER], in which case control is passed back to the DEMOS file manager.

To force **Hunter 2** to try again, press the X key. This will force transmission for ETX/ACK or XON/XOFF and send the buffer three more times in the case of ACK/NAK.

To prevent a timeout after a transmission failure, re-initialise communications.

The communications error messages are as follows.

If you see an error message that is in a different form to this, it is not a communications error message! For example, it might be a message from the user program or remote system. Such errors are of course beyond the scope of this manual.

Error 01: Unanswered line bid

This is due to not receiving any reply in a request for the line, usually caused by incorrect connection of the communications channel. (Synchronous mode only).

Error 02: No response from receiver

After establishing the line, a buffer has been sent to which no response has been given, despite 10 ENQs having been sent. (Synchronous mode only).

Error 03: Response not matched by odd-even block count

The wrong positive acknowledgement has been given to a buffer, despite confirmation requests. (Synchronous mode only)

Error 04: Retransmission rejected

Repeated transmission of a buffer gets only the NAK response, has tried 10 times. (Synchronous mode only).

Error 05: Loss of response from receiver

Having sent out a WACK no acknowledging ENQ has been received within the 5 second timeout. (Synchronous mode only)

Error 06: Not used

Error 07: Not used

Error 08: Receive timeout Error

No data has been received to a request for input (eg LINPUT) for the specified time.

Error 09: Transmit timeout Error

Transmission has not been able to take place in the specified time.

Error 10: ETX/ACK timeout Error

No response has been received for 30 seconds.

Error 11: ACK/NAK failure

Three attempts have failed to get an ACK.

Error 12: Systime failure

Ten re-transmissions have failed to get an ACK.

Chapter 8 Looking After Hunter 2

Hunter 2 is very robust and has been designed for a long and trouble-free useful life. However, like any precision instrument, it will only give its best if treated with a certain amount of care.

This chapter summarises the few simple rules that you should follow to ensure that **Hunter 2** gives you many years of trouble-free service.

8.1 Everyday Use

In everyday use, all you need to do is treat **Hunter 2** with a little respect:

* press the keys gently but firmly
* don't lay heavy items on top of it
* don't leave the display exposed to bright sunlight for long periods
* if a low power warning is displayed, turn **Hunter 2** off and renew or recharge the batteries as soon as convenient

Hunter 2 is designed for operation in normal indoor and outdoor environments. For more details, see Appendix II.

8.2 Case Sealing

All **Hunter 2**s are built with integral seals to provide a high degree of protection against atmospheric moisture, corrosive gases and accidental immersion, therefore **Hunter 2** should not be opened.

The battery seal is a small 'O' ring seal running in a machined groove in the battery plug and seating on a machined face in the **Hunter 2** casing. It requires a smear of silicon grease to assist in seating.

Under certain circumstances (change of altitude, etc.) the interior of **Hunter 2** may develop a substantial pressure difference compared with the environment.

If there is internal overpressure, the keys will become stiff and difficult to operate. If there is external overpressure, **Hunter 2** will show a tendency to ingest any rainwater adjacent to the seals, and in extreme cases, may activate its keys.

In either case, release any pressure differential by loosening the battery cap.

If you are taking **Hunter 2** on an airline trip, this is a sensible precaution to take.

8.3 Humidity Indicator

Hunter 2's interior is kept dry by a silica gel dessicant, in a muslin bag. The dryness of **Hunter 2**'s interior is essential to its correct operation, and is indicated by a humidity indicator, visible through a small hole in the top left corner of the display window.

Check the colour of the indicator at intervals.

THE HUMIDITY INDICATOR MUST ALWAYS BE BLUE

If it takes on a pinkish colour, the unit must be returned to HUSKY Computers or an authorised Service Agent promptly to avoid corrosion of internal parts. If the colour changes immediately after an accident involving water or high humidity, a fault has occurred in the sealing, and there could be water inside the unit.

If you suspect that water has entered the interior, you MUST take the following action:

1) Remove the batteries.
2) Return the **Hunter 2** to HUSKY Computers, or authorised Service Agent.

8.4 Storage

If storing **Hunter 2**, observe the storage conditions described in Appendix II.

If you wish **Hunter 2** to retain the data stored on its internal disk, store it with its batteries fitted and check these at regular intervals (typically once a month), for charge and any signs of leakage. If the batteries are deteriorating, replace them at once.

Chapter 9 Accessories

9.1 Introduction

Hunter 2 uses an operating system, protocols and communications interface that are compatible with the industry standards. This makes it easy for you to extend its capabilities, by adding a variety of standard peripheral devices.

Hunter 2's potential for software expansion is also considerable. By providing a CP/M-compatible environment, it allows you to use any of the enormous range of commercial CP/M software, such as wordprocessors and spreadsheets.

Details of these standard peripherals and programs should be available from your supplier

Husky Computers have also developed a range of their own peripherals and software enhancements for the **Hunter 2**, such as portable printers, disk drives and card readers, barcode readers and extra communications protocols.

This chapter briefly describes the main accessories developed by Husky Computers. For more details, consult the product user guides or approach your local distributor or Husky Computers.

9.2 Barcodes and Light Pens

9.2.1 Introduction

As an option, **Hunter 2** is capable of reading three types of barcode:

- Code 39
- EAN 8/13
- Interleaved 2-of-5

To provide these facilities, software packages are loaded into the file system of **Hunter 2** as permanent files and accessed by WAND input commands such as WINPUT and WINCHR (in BASIC). User access to these files is not allowed.

If an attempt is made to access these files without them being present, then a "system file error" will be displayed.

To check the version number of your barcode software, type the following in the File Manager·

`TYPE` *filename* ENTER

(This is not available for versions earlier than 9G08.)

Software to read other barcodes is available to special order If you require assistance to implement a barcode system application, please contact Husky Computers for Technical Note 231.

See Part 5 for details of connecting and removing barcode wands.

9.2.2 Using a Barcode Wand

There are a few simple rules to follow when using a hand held wand:

If dust builds up on the wand lens, it will gradually reduce reliability of wand operation. To avoid this, once a week, unscrew the tip and check for dust. If any dust is found, wipe it off gently with a soft cloth.

Move the wand lightly across the barcode. Do press hard or the tip may wear away, altering the focal length of the lens and reducing reliability

Move the wand across the barcode at a constant speed, maintaining contact with the barcode. The scan should be at right angles to the bars of the barcode.

Hunter 2 will tolerate a considerable variation in scanning technique. The scan can be slow, fast, traverse across the barcode in an arc rather than a straight line, or even in a 'wavy' line, provided that the wand remains within the barcode region. You can also scan the barcode in either direction.

It is not even necessary to place the wand precisely at the beginning of the barcode before a scan since **Hunter 2** will work out which information transmitted from the wand actually corresponds to the barcode itself. If the wand is resting on a barcode, it is possible to move the wand to either end of the barcode and then scan in the opposite direction. **Hunter 2** will then work out the beginning and end of the barcode.

Hunter 2 will allow a maximum of 25 characters to be represented by a Code 39 barcode. Bear this in mind if you intend to produce your own Code 39 barcode labels. If larger barcodes are required, contact Husky Computers for details.

9.3 Battery Charger

THESE INSTRUCTIONS RELATE TO PERSONNEL SAFETY AS WELL AS RELIABLE OPERATION OF HUNTER AND BATTERY CHARGER. IT IS IMPORTANT THAT THEY ARE READ AND UNDERSTOOD.

9.3.1 Description

Hunter 2's rechargeable battery system consists of four rechargeable cells and a HUSKY **Hunter 2** battery charger.

The cells are 'AA' size Nickel Cadmium (NiCad) of nominally 1.2 volts, 0.50 Ah and are capable of being charged at 65mA continually.

The special HUSKY **Hunter 2** battery charger is the only unit which may be used to recharge the NiCad batteries when they are installed in **Hunter 2.** The use of an ordinary AC adaptor/battery eliminator may damage **Hunter 2** or the batteries.

9.3.2 WARNING

When using the charger be absolutely certain that **Hunter 2** has four NiCad cells correctly aligned and no other type is present. Do not even mix NiCads of different age or state of charge, as this produces a risk of chemical leakage, gassing or explosion. Do not attempt charging at temperatures of 5 degrees C or less.

The charger contains lethal voltages, so under no circumstances should it be opened, in any way tampered with, or used for any other purpose. It must, of course, be kept dry There are no user serviceable items inside the case.

If you suspect that the charger is faulty, you may change the fuse inside the mains plug for a similar fuse not exceeding 2 amps rating. The charger unit is fully protected against continuous short circuits but if unlikely fault conditions arise which cause overheating of the charger, it will self-destruct, quietly and safely. The charger is double insulated, making no earth connection.

9.3.3 Operation

To use the charger unit the following procedure is recommended:

1) Switch off **Hunter 2.**
2) Plug the charger into **Hunter 2**'s LEMO connector.
3) Plug the charger into the mains.
4) Finally switch mains on.

See Part 5 for details of connecting and removing the LEMO plug.

Hunter 2 operation is possible during use of the charger but the current drain will counteract the charging current and consequently prolong the time to fully charge the batteries. Removal of the cells under this condition is forbidden.

As with all electrical apparatus the charger should be disconnected from the mains when not in use. Only 240V AC (220 to 250V) mains, 50 or 60 Hz is suitable for standard units. Chargers for alternative mains supplies, eg 110V, are available.

Exhausted batteries will typically be fully charged in 12 hours, overnight charging is a popular practice.

9.3.4 Replacing the Rechargeable Batteries

It must not be forgotten that rechargeable batteries do not have an unlimited life and will ultimately need replacing as do car batteries. If the batteries become exhausted after little use then the set may need replacing. Their life expectancy can be several years but this is reduced by repeated or prolonged total discharge and by excessively high temperatures.

NOTE: Several discharge/charge cycles are required before NiCad cells reach their peak capacity. This type of battery will also self discharge, especially at elevated temperatures, which can result in a fully charged set of batteries becoming flat in a few weeks. An occasional charging session, say, every fortnight is recommended if **Hunter 2** is being stored.

9.3.5 Mains Operation

For critical or remote operations, permanent charging may be called for This is acceptable provided ventilation around the charger and **Hunter 2** is good, but ageing of the cells may be accelerated. Annual replacement would be a prudent action in this instance.

Refer to Part 5 for more details.

9.4 Husky Oracle GT Disk Drive

9.4.1 Operation of the Oracle GT

The **Oracle GT** disk drive provides far superior facilities to the previous Oracle disk drive. It allows you to use disks which are compatible with the standard IBM 3.5 inch format, with either 720K byte or 1.4M byte capacity. These disks provide a convenient means of exchanging files between the **Hunter 2** and many desktop personal computers, as well as providing **Hunter 2** with a very large capacity back-up media.

Note, however, that the disk format is not compatible with that of the previous Oracle disk drive. You cannot swap disks between the two drive types.

Always use good quality disks, particularly for the 1.4M capacity format. Using inferior disks is a false economy, as it will lead to data loss and file corruption.

Powering Up

To power up the **Oracle GT** either:

1) plug in a suitable charger

> **WARNING!**
>
> Only use a suitable Husky charger. Use of unauthorised chargers may damage the unit or cause injury.

2) Operate the power switch on the rear panel

On power-up your **Oracle GT** will activate the disk drive, home the head, and also momentarily illuminate all LED. The latter facility allows the user to quickly identify any LED failures should they occur.

The LED display then displays the battery status on the four green LEDs, and the status of the charger with the red LED.

The **Oracle GT** is now ready to receive commands.

Reasons for failure to power on:

1) If the unit fails to latch on the power when the power switch is operated, insert a charger as the battery may be low.

2) If the unit fails to latch on power when the charger is inserted attempt to reset the unit using the configuration mode as described later in this chapter

Power Management

The **Oracle GT** operating system utilises several means to reduce the overall power consumption to a minimum.

1) The power to the drive is automatically shut off if no disk operation occurs within several seconds.

2) If no communications are received for approximately 30 seconds by the **Oracle GT** the unit will power off completely unless:

   i) an active device is connected to the **Oracle GT** port

   or

   ii) the charger is inserted

The rechargeable Ni-Cad battery fitted in the **Oracle GT** should provide many hours of operation. The front panel display gives an immediate indication of the battery status, and also the status of the charger:

The green power LEDs illuminate from left to right as the power increases. For example 75% capacity is shown like this:

When the battery capacity is very low the 25% LED will flash approximately once per second. If this happens, recharge the battery as soon as possible or the drive may turn itself off during a disk access.

Battery Charging

The **Oracle GT** operates an intelligent battery charging system. When the charger is inserted and the battery is low, a rapid charging circuit is activated to quickly charge the battery. During this period the red charger LED flickers approximately once per second.

When the battery is fully charged, the software switches in a trickle charging circuit. If the drain on the battery is large the software will automatically switch in the rapid charging circuit if the battery capacity falls below 90% and the charger is in. When the battery is fully charged the red charger LED flashes at a rate of approximately 2Hz, and remains in this state until the charger is removed.

Plugging the charger into an **Oracle GT** when it is powered off will cause it to power up.

WARNING!

Only use a suitable Husky charger. Use of unauthorised chargers may damage the unit or cause injury.

As charging progresses, the power capacity will be shown on the green power LEDs.

NOTE:

The unit will be fully charged when both the charging LED flashes, and the LEDs indicate 100%. This event should occur in approximately one hour of charging time. Failure of this may indicate a fault in the charging circuit. It is advisable to contact Husky Computers Limited or their representative for advice should this occur

The unit's battery capacity should be maintained by regular charging. When the battery capacity is extremely low the 25% LED will flash approximately once per second indicating charging is needed. Failure to charge the unit will cause it to turn off due to low power

If more convenient, you may safely operate the disk drive with the charger continually connected.

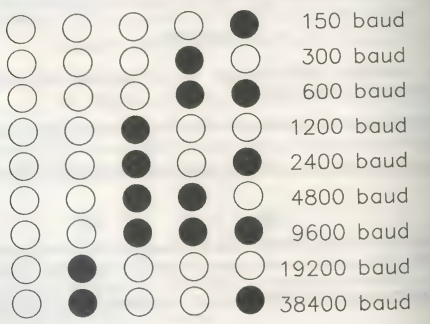
Baud Rate Selection

To use the **Oracle GT** drive with **Hunter 2**, you must first set the Baud rate to 4800 Baud, as follows.

The baud rate configuration mode may be entered by operating and holding in the power-on switch on your **Oracle GT** disk drive only when the charger is inserted. This prevents inadvertent alteration of the drive settings.

Your **Oracle GT** will detect the presence of the power key provided the switch is held in for approximately five seconds and the charger is inserted. When detected the unit will indicate that configuration mode has been entered by flashing all the LEDs simultaneously.

When the power key is released, the unit will then change to a flashing LED display representing the current baud rate setting:



To increment the baud rate, press the power key.

When the desired baud rate is displayed, leave the unit for approximately five seconds, and it will store the new rate in its memory.

On power-up for the first time, or after removal of the battery, the baud rate setting will default to 19200 baud.

IMPORTANT!

Setting the baud rate also causes the battery capacity to be forced to 25% capacity. This allows recalibration of the battery monitoring system.

9.4.2 Setting up

Introduction

The Oracle GT can operate directly with **Hunter 2** without the need of any modifications. This section provides operating instructions for use of the drive in this situation.

The **Oracle GT** System disk provided with your **Oracle GT** contains important files for use with the unit and Husky products. Prior to utilising the disk ensure that the write protect tab is open to prevent inadvertent writing onto this disk.

It is recommended that a copy of this disk be made at the earliest opportunity, and the original kept safe for future reference if necessary.

It is intended that this mode is for convenience only, enabling system files to be down loaded from the system disk provided with the **Oracle GT**.

There are several restrictions which apply:

1. The file space area which is accessible on the disk is restricted to a maximum of 100K bytes.
2. The disk is effectively read-only, no files may be stored to the disk.
3. The disk may not be FORMATTED
4. The baud rate must be set to 4800 baud for operation with the Hunter.

The disk to be read must be either an IBM 3.5 inch compatible 720K or 1.4M disk format.

IMPORTANT!

The **Oracle GT** CANNOT read the original Oracle FB100 single sided disks.

These disks are not in a compatible form which can be read by the disk unit.

Setting up using HDOS.COM

Users who already have access to the HDOS.COM program for the old Oracle Disk Drive may proceed as follows in order to load the DSK.COM program from the new **Oracle GT**

Users who have received the Oracle for the first time should refer to the next section, which describes how to use the DISKBOOT program.

1. Connect the disk drive to the Hunter using the cable provided. Ensure the disk drive is powered on by either:

   Operating momentarily the push button on the rear of the drive unit.

   or

   Inserting the charger into the unit.

2. Insert the **Oracle GT** system disk or other suitably formatted disk.
3. On the Hunter in DEMOS type the command HDOS and use the LOAD facility to read the file DSK.COM from the system disk.

This should copy the **Oracle GT** control file from the system disk to the Hunter RAM disk. This file will be used in place of HDOS.COM control file used with the previous Oracle disk drive.

Failure Modes

The following is a summary of potential problems which may be encountered and possible causes.

* Disk drive fails to power on.

  If drive fails to 'latch on' when the rear push button is operated it suggests low power - insert the charger

* Rubbish or no data returned in response to DIR.

Incompatible or non-formatted disk in the drive - check the disk or use an alternative disk.

* No response at all from disk.

  Check the **Oracle GT** baud rate setting as explained in Chapter 2.

Setting up using the Husky Hunter Disk Boot Program

Hunter users who have received their **Oracle GT** for the first time must load the DSK.COM software utility from the system disk to utilise the drive, as follows:

1. Set the Hunter Communications Parameters as follows:

```
Transmission Parameters
Rate-4800  Prtcl-none      Pty-8 bit
CTS-y DTR-n LF-n Echo-n T/O-no Null- 0
press ENTER if acceptable.
```

```
Receiving Parameters
Rate-4800  Prtcl-none      Pty-8 bit
RTS-hold DSR-n DCD-n  T/O-no Serig-off
press ENTER if acceptable
```

2. Connect the **Oracle GT** disk drive to the Hunter, and insert the system disk supplied into the drive.

3. Enter the following Hunter BASIC program into the Hunter and save it as DISKBOOT.HBA.


```
10 REM DISKBOOT V1.10
20 A=ARG(38) : A=CALL(5)
30 A=40000 : T=49 : S1=48 : S2=54 : C=ARG(3)
40 FOR A=1 TO 24
45 PRINT ".";
50 IF A=4 OR A=14 OR A=24 THEN S1=49 ELSE S1=48
60 IF A=5 THEN T=50
70 IF A=15 THEN T=51
80 S2=S2+1 : IF S2=58 THEN S2=48
90 LOPCHR 82, T, 44, S1, S2, 13
100 FOR B=0 TO 7 : C=CALL(5)
110 IF C <> 48 THEN PRINT "ERROR":END
120 NEXT : LOPCHR 13
130 FOR P=P TO P+127
140 POKE P, CALL(5)
150 NEXT : NEXT
160 BEEP : A=CALL(40003)
```

4. Power up the **Oracle GT** disk drive and ensure the baud rate is set the 4800 baud as described in the previous section.

5. Execute the program by typing RUN.

The DISKBOOT program will read the relevant sectors off the system disk in which the DSK.COM utility resides and store the program in the TPA.

> **IMPORTANT!**
>
> DISKBOOT will only function correctly with the system disk or an identical copy of it.

When the program is loaded completely the DISKBOOT program issues a beep and returns to DEMOS.

6. Once in DEMOS type the following to save DSK.COM:

   `SAVE 12 DSK.COM` ENTER

When you are sure that DSK.COM operates satisfactorily, you may erase the DISKBOOT.HBA program.

If DISKBOOT.HBA displays an error message, check:

a) The disk drive is connected and powered up.

b) The baud rate is correctly set.

c) A system disk is in the drive.

d) The communications on the Hunter are correct.

e) You have typed in the DISKBOOT.HBA program correctly.

9.4.3 Using the Oracle GT with Hunter 2

Introduction

To use the **Oracle GT** with the Husky **Hunter 2** you will need the **Oracle GT** charger· you cannot use the **Hunter 2** charger.

The user interface to the **Oracle GT** is provided by the DSK.COM utility, which must be loaded onto the RAM disk from the system disk, as described in the previous section. This utility allows you to transfer files between the drive and the **Hunter 2**, and also provides commands for maintenance and creation of disks. The command options available are described in the remainder of this section.

For details of using the Oracle GT from applications, see Appendix VI.

DSK DIR filename

This command provides a directory of the disk contained in the Oracle GT together with the parameters relating to the files. The filename may contain wildcards.

For example:

```
DSK DIR *.BAS [ENTER]
DIR *.BAS

          Recs  Bytes   Acc
CHECK BAS    2     2K   R/W
TEST.BAS     1     2K   R/W

R/W 306K bytes available
```

The above commands gives the directory status of all .BAS files on the **Oracle GT** disk.

DSK DUMP filename

Copy files from the RAM disk to the **Oracle GT** Disk. The filename may contain wildcards. As the files are copied their names appear on the screen.

For example:

```
DSK DUMP *.COM [ENTER]
DUMP *.COM
TEXT.COM
WBOVLY1.COM
BASIC.COM
>
```

The above command copies all .COM files from the RAM disk on the **Hunter 2** to the **Oracle GT** disk

DSK ERA filename

Erase the file explicitly specified in the filename. For example:

```
DSK ERA HELLO.COM [ENTER]
```

```
ERA HELLO.COM
>
```

The above command erases the file HELLO.COM from the Oracle GT disk.

DSK REN new name = old name

Rename the file *oldname* on the Oracle GT to the filename *newname*. The file names specified must be explicit with no wildcards. For example:

DSK REN new.com = old.com ENTER

```
REN NEW.COM = OLD.COM
>
```

The above command renames the file OLD.COM to NEW.COM on the Oracle GT disk.

DSK FORMAT

Format the disk in the **Oracle GT** For example:

DSK FORMAT ENTER

The above command causes the following screen to appear:

```
Oracle GT Format

enter 1 for 1.4M or 2 for 720K disk
```

On selecting the disk type the screen changes to display:

```
Oracle GT Format

press enter to begin or ^C to abort
```

On pressing ENTER the format will commence and the screen will display progress, like this:

```
Oracle GT Format

press enter to begin or ^C to abort

FORMATTING 1.4M disk
```


The clicking of the stepper motor on the drive should be clearly audible if format is in progress. The command can be aborted using CTL/FN C.

Once completed a message will be displayed to indicate success or failure.

DSK LOAD filename

Copy files from the **Oracle GT** disk to the **Hunter 2** RAM disk. The filename may contain wildcards. As the files are copied their names will be displayed.

For example:

DSK LOAD *.COM ENTER

```
LOAD *.COM
TEXT.COM
MSOVLY1.COM
BASIC.COM
>
```

The above command copies all .COM files from the Oracle GT to the **Hunter 2** RAM disk.

Error Messages

The DSK.COM utility will generate error messages should difficulties occur while performing the command.

Aborted

Use of CTL/FN C to abort the function will usually result in this message.

Cannot make file

A failure has occurred in creating a destination file. This could be due to either insufficient directory space or the destination drive is set to read only

Cannot open file

Access to a file has been refused. This is usually due to the an error in the file itself.

Disk error

A timeout has occurred in the disk access. Check that the connections are correct, a disk is in the drive and the baud rate is set to 4800 baud on the drive.

File read only

An attempt has been made to delete or write to a file with its read only attribute set.

No file

A search has failed to detect the file required.

Syntax

The syntax error message is usually accompanied by the command line input. A syntax error could be due to either·

1 an invalid command
2 invalid command syntax
3 invalid characters in filenames
4 use of wildcards when prohibited.

HUSKY HUNTER 2 TM

Part 3
Hunter Basic

Chapter 10 Introduction to Hunter BASIC

10.1 Introduction

Hunter BASIC is a powerful and flexible BASIC Interpreter included within **Hunter 2**'s firmware, to allow you to write and execute programs without any need for additional equipment. In addition to the features expected of a modern microcomputer BASIC, **Hunter** BASIC can also handle communications with other devices.

Hunter BASIC also has a number of special features designed for use in the portable environment. Principle among these is its ability to preserve the current program and data after power has been switched off. Data can be used later by the same program or another program, even though **Hunter 2** has been powered down and not used at all. Similarly, user programs written in **Hunter** BASIC are retained and are always available for further use until deliberately cleared.

Different programs can be stored simultaneously in program space or in files and executed independently.

For information on writing machine code routines and assembly language applications, see Part 4. Part 4 also describes memory locations and ports that can be used to control **Hunter 2** at a low level, from BASIC. For details of using machine code routines from BASIC, see section 10.11.

A number of example programs can be found in Appendix VI.

10.1.1 Selecting Hunter BASIC

Hunter BASIC is accessed from the File Manager by typing:

BAS [ENTER]

or

[CTL/FN] 3

When the BASIC Interpreter has been entered the following screen is displayed:

```
Hunter Basic Interpreter
READY


File  Run Edit List Save Load Kill Syst
```

You will be able to use the last BASIC program used, together with its variables, as long as you have not used a utility which writes to the TPA (for example, INP or the editor) since then.

To return to File Manager from BASIC, type:

CTL/FN 8
or
SYSTEM ENTER

Alternatively, you may execute a **Hunter** BASIC program (.HBA extension) directly from the File Manager by typing its filename followed by ENTER. In this case, the BASIC Interpreter screen does not appear and the program executes immediately. Programs may also be executed by using the CONT and /DO commands, with somewhat different results, as described in Part 2.

10.1.2 BASIC Programs

Program source code for execution by the resident BASIC Interpreter can be provided by a variety of sources, including the following:

1) **Hunter** BASIC (.HBA extension) files.
2) Text (TXT extension) files.
3) Direct keyboard entry using Edit.
4) Down line loading using LLOAD.

Programs are entered into **Hunter 2**'s execution memory as a sequence of line numbers and program statements. Lines without numbers will execute immediately Variables can always be inspected using the PRINT command. The effects of most commands can be determined empirically by simply typing the command and observing the result, although certain commands (eg GOSUB) may not be executed directly

Type NEW before entering a program, to remove any previous program.

10.1.3 Autostart BASIC Programs

Provision has been made for BASIC programs to execute immediately on power up. See section 10.15.

10.2 BASIC Syntax

The keywords described in this part of the manual follow the conventions set out below. The syntax of all BASIC statements and commands must be correct for the program to run successfully

- BASIC reserved words are printed in capitals, but they may be entered as capitals or lowercase or even a combination of both. Either way, the Interpreter will convert them to capitals.
- Items not in square brackets are essential arguments to the BASIC function and must be included.
- Square brackets indicate that the item is optional.
- Punctuation must be included where shown.

10.2.1 Types of Keyword

Hunter BASIC understands four kinds of keyword:

Commands	generally initiate an action and can also control execution of application programs.
Statements	form the structure of application programs and direct program flow
Operators	are used to relate data and to test for specified conditions.
Functions	perform arithmetic or string computations, returning a numeric or string result.

Some commands and statements will function without further information, others require an argument. An argument is a constant, expression or function of the appropriate type.

10.2.2 Expressions and Operators

There are four types of expression: numeric, string, logical and relational.

Numeric expressions and operators

A numeric expression is defined as *numeric* [*operator numeric*] where a *numeric* is a numeric constant, variable, function or expression and *operator* is one of the items in the following table and the portion in square brackets may be omitted or repeated a number of times.

A numeric expression is evaluated to yield a single numeric result (ie a number), when the instruction containing the expression is executed. For a description of the rules for evaluating an expression, see section 10.2.3. The way that a numeric expression is evaluated may be controlled by the uses of parentheses: "(" and ")"

+	Addition
-	Subtraction or negation of value
*	Multiplication
^ or **	Powers
/	Division
=	Equality or assignment of values
= >	Equal to or greater than
= <	Equal to or less than
<	Less than
>	Greater than
< >	Not equal to

While the last six operators may be used in numeric expressions, they are more often used in relational expressions (described below). The = operator is also used to assign values to variables.

String expressions and operators

A string expression is defined as *string* [+ *string*] where *string* is a string constant, variable, function or expression and the portion in the square brackets may be omitted or repeated any number of times. The + operator concatenates strings, ie adds the string to its right onto the end of the string to its left. The result is a single string.

A string expression is always evaluated from left to right.

Relational expressions and operators

A relational expression takes the form *string relational-operator string* or *numeric relational-operator numeric*. It compares two values of the same type (ie both string or both numeric), yielding a numeric result indicating whether the relationship between them is as stated by the relational operator. A result of -1 indicates that the relationship is true, 0 indicates that it is false.

=	Equality or assignment of values
< >	Not equal to

Relational expressions are most often used as the controlling condition in an IF statement or WHILE loop.

Logical expressions and operators

Logical operators perform logical or Boolean operations on numeric values, as follows.

=	Equality or assignment of values
= >	Equal to or greater than
= <	Equal to or less than
<	Less than
>	Greater than
< >	Not equal to

NOT	logical complement
AND	conjunction
OR	disjunction
XOR	exclusive OR
EQV	equivalence
IMP	implication

NOT	X	NOT X
	T	F
	F	T

AND	X	Y	X AND Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F

OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F

XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F

EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T

IMP	X	Y	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T

Logical operators act upon the binary equivalents of the numeric arguments, bit by bit, where the argument is interpreted as a 16 bit signed 2's complement number ie in the range -32768 to +32767 For example:

```
PRINT 10 AND 6
```

will print 2, since:

10 is 0000000000001010
and
6 is 0000000000000110
so 10 AND 6 is:
0000000000000010, printed as 2 in decimal

Logical operations may also be used by conditional statements, for example:

```
IF A=B AND B=C THEN PRINT"ALL EQUAL"
```

In this example the following occurs:

1) The variables A and B are tested for equality If they are equal then a true (-1) condition is returned, if not then a false (0) is returned.

2) The variables B and C are tested for equality The results are as in (1) above.

3) The logical operation AND is then performed on the results of (1) and (2) above, which returns either a true (-1) or false (0) result. It is this final result upon which the IF statement performs its conditional test (true or false).

Since conditional statements always produce a logical result it is possible in some circumstances to omit a relational operator, for example:

```
IF A<>0 THEN PRINT"A IS NON ZERO"
```

may be replaced with:

```
IF A THEN PRINT"A IS NON ZERO"
```

Similarly·

```
IF A=5 THEN PRINT -1 ELSE PRINT 0
```

may be replaced with:

```
PRINT A=5
```

since A and 5 are tested for equality and if they are equal the -1 (true) is printed otherwise 0 (false) is printed.

10.2.3 Operator Precedence

If an expression contains more than one operator, the result depends on the order in which the operators are applied. This is called 'operator precedence'

An expression is evaluated by applying the highest precedence operators first, to give intermediate results, then the next highest operators, and so on. Operators of equal precedence are applied sequentially, from the start of the expression (ie left to right). Parentheses can also be used to force parts of the expression to be evaluated independently

The operator precedence for **Hunter** BASIC and the industry standard Microsoft BASIC is as follows:

Microsoft	**Hunter**
^ (negation)	NOT - (negation)
* /	^ * /
+ (subtraction)	+ - (subtraction)
NOT	AND
AND	OR
OR	XOR IMP EQV
XOR	
IMP	
EQV	

This means that **Hunter** BASIC and Microsoft BASIC may yield different results for the same expression. For example, consider the expression:

```
0.5 * (30-20) ^3
```

Microsoft BASIC will evaluate the expression within the parentheses first:

ie: `0.5 * (30-20)^3 = 0.5 * 10^3`

The exponentiation operator, ^, has a higher priority than the multiplication operator, *, so the next stage is:

```
0.5 * 10^3 = 0.5 * 1000
```

and the result is: `0.5 * (30-20)^3 = 500`

Hunter BASIC will evaluate the expression within the parentheses first:

ie: `0.5 * (30-30)^3 = 0.5 * 10^3`

However, unlike Microsoft BASIC, the multiplication and exponentiation operators have identical precedence and, therefore, the expression is evaluated from left to right:

ie: `0.5 * 10^3 = 5^3`

and the result is: `0.5 * (30-20)^3 = 125`

10.3 Program Lines

10.3.1 Line Numbers

Line Numbers are in the range 0 to 65535. Each line can be up to 251 characters long, including the line number Extra lines can be inserted in the text by simply typing in a new line number between the two lines of interest. In view of this, it is recommended that line numbers are separated by an increment of 10 for each new line. Line numbers are stored in binary form and occupy a constant space in memory regardless of the number chosen. Each line number requires 2 bytes of memory.

There is no renumber facility in **Hunter** BASIC.

10.3.2 Multi-statement Lines

Hunter BASIC supports multiple statements in a single program line. The statements must be separated by colons (:).

For example:

```
100 A=0:B=10:PRINT A,B:B=11
```

The primary use of this feature is to reduce the program size. It may also help to make a program more legible.

NOTE: If a colon is used at the end of a line, END is assumed and program execution stops.

The following keywords cannot be used in a multi-statement line:

DATA
WHILE
WEND

If REM is used in a multi-statement line, it must be the last statement in the line.

10.4 Types of Variables

The BASIC workspace execution memory is automatically partitioned by the Interpreter according to the number of program lines entered by the user and the variable storage space required. If more than 54K RAM is required then data must be stored as files or the program may be suitable for execution from the filespace, leaving the entire TPA free for storage of numeric and string arrays.

10.4.1 Numeric Variables

All simple numeric variables are single precision giving 6 digit accuracy. Variables are designated by one alphabetic character (A-Z) followed by an optional alphanumeric character (A-Z or 0-9), giving up to 962 separate variables.

NOTE: Simple variable names ON, TO, IF, LN, AS, OR and PI should be avoided as these are also reserved words. Use as variables can lead to execution errors.

10.4.2 Numeric Arrays

Numeric variables may form a one-dimension array using "DIM".

Single Precision

Single precision arrays are designated by a number in parentheses following the variable name, like this:

```
A1(19),Z(2),C(X),A(20*T),AD(10),AZ(62)
```

There can be 962 separately identified arrays of single precision.

NOTE: The number in parentheses can be a numeric expression. Each variable occupies 5 bytes of **Hunter 2**'s user memory in arrays, 7 bytes otherwise.

Double Precision

Double precision arrays give 14 digit accuracy but with an additional memory overhead. Double Precision arrays are identified by the symbol '!' followed by a number or expression in parentheses.

```
eg A1!(19),Z!(Z),C!(X),AA!(23)
```

These are a further 962 arrays, distinct from single precision arrays.

Each double precision element occupies 9 bytes of **Hunter 2** memory.

Default Values

All numeric arrays default to 11 elements unless dimensioned with DIM statement.

10.4.3 String Variables

Single element string arrays are automatically assigned without a DIM statement. The default length is 20 characters.

String names are defined exactly as variable names, with one alphabetic character (A-Z) and one optional alphanumeric character (A-Z or 0-9) followed by the '$' symbol giving a total of 962 separately identified string variables.

Hunter supports strings of 8 bit characters including the full ASCII character set.

The maximum length of any string is 254 characters, with a default size of 20 characters.

```
DIM A1$(10,15)
```

defines a string array of 11 elements, each of which can be up to 15 characters long.

The maximum size of an array depends on the memory available. Examples of String variables are:

```
A1$,Z$,C5$(22),H$(J),MM$(32)
```

String variables can be initialised by assigning using = (and using the optional LET keyword), or by using the READ command to read from DATA statements.

The first element of a string array may be referenced with just the variable name, without the need for any number in parentheses.

For example:

A1$ and Z$ actually refer to the variables A1$(0) and Z$(0)

A variable might therefore be defined as ST$(0,10) and used as ST$.

10.5 Variable Structure

Note: in the following diagrams indicating memory usage, the leftmost element is stored first in memory and the memory locations increase reading from left to right.

10.5.1 Simple Numeric Variables

A numeric variable has a 2 byte entry in the symbol table, with 5 bytes used for the value: 6 BCD digits (3 bytes), a sign byte, and a mantissa byte:

d1	*d2*	*d3*	*s*	*m*

Where *d1* is the two most significant digits and *d3* gives the two least significant digits. The sign byte *s* is 00H if the number is positive, 01H if the number is negative. The mantissa byte is as follows: if the number is considered as 0.nnnnnn * 10^p, *m* is 80H + p.

VARPTR returns the address of the mantissa, *m*.

10.5.2 Numeric Array Variables

A numeric array has a 7 byte entry in the symbol table, plus 5 bytes for each element, structured exactly as described for a numeric variable. These elements are stored consecutively in memory, element 0 first.

VARPTR returns the address of the mantissa of the element specified as its argument.

10.5.3 Double Precision Array Variables

A double precision array has a 7 byte entry in the symbol table, plus 9 bytes for each element, structured rather like a numeric array element:

d1	*d2*	*d3*	*d4*	*d5*	*d6*	*d7*	*s*	*m*

Where *d1* to *d7* are the 14 BCD digits, *s* is the sign and *m* is the mantissa.

The array elements are stored consecutively in memory, element 0 first.

VARPTR returns the address of the mantissa of the element specified as its argument.

10.5.4 String Variables

A string variable has a 7 byte entry in the symbol table, plus an entry for each element, structured like this:

len	*c1*	*c2*	*c3*	*c4*	*etc*

Where *len* is the current length and *c1*, *c2* etc are the ASCII characters.

VARPTR returns the address of *c1*. For example:

```
X=VARPTR(A$(5))
```

will return the address of the first character of A$(5) to X. The second character is stored at X+1, third character at X+2, and so on. The length of the string is stored at X-1.

10.6 Editor

A text editor is provided for creating and modifying text and BASIC program files. It may also be used to modify the current BASIC program.

To edit the current program, type:

`EDIT` ENTER

or

CTL/FN 3 ENTER

To edit or create a different file, type:

`EDIT "`*filename*`"` ENTER

or

CTL/FN 3 "*filename*" ENTER

To return to BASIC, select the EXIT function key from the editor. This will convert the text to the compressed form used by BASIC and leave it in memory as the current program.

The editor provides a wide range of facilities, described in Part 2.

10.7 Keyboard

10.7.1 Introduction

Hunter 2 computers are supplied with a standard QWERTY keyboard as described in Part 1.

10.7.2 Redefining the Keyboard

Keyboard keys are each equated to the characters they represent by unique numerical codes, as listed in Appendix III. These values are defined according to the American Standard Code for Information Interchange (ASCII).

The value returned to BASIC in, for instance, an INCHR statement is the numerical equivalent of the key pressed.

The ASCII definitions of each key for the keyboard are contained in 112 consecutive RAM locations named KEYBUF The upper shift keys are defined in the first 56 locations, followed by the 56 lower shift keys. These memory locations form a 'map' of the physical keyboard. On power up, the **Hunter 2** automatically defines these locations to a standard configuration. However, the programmer has the option to be able to define special keyboard arrangements.

For full details of the key locations and programming special actions, see 'Redefining the keyboard', in Part 4.

As an example, consider the key fourth from the left on the top row To program this key to be $ in upper case and S in lower case, two POKE operations would be required:

```
POKE KYBUF+6,36
POKE KYBUF+62,83
where KYBUF is the keyboard buffer ram
```

When **Hunter 2** powers up, the keyboard will revert to the standard version unless in 'CONT' mode (for example, after executing an ON POWER, RESUME command). The program must reprogram the special keyboard requirements each time the program is run.

The keyboard can be reprogrammed to simplify certain operations. For example, on the **Hunter** SP model, the CTL/FN key is a shifted key, so to type a control code, you need to press three keys! To avoid this problem, you could temporarily make the ㉮ key act as the CTL/FN key:


```
1  SK=65320:CK=65306
10 POKE SK,132
20 POKE (SK+56),132:REM Relocate control key
30 POKE CK,0
40 POKE (CK+56),0:REM   Disable old control key
50 POKE 65378,0:REM     Change layout
```

Once this has been run, ㉮ will act as a CTL/FN key, allowing users to type control functions by depressing just two keys.

After use, the values should be returned to their original states, ie:

```
100 POKE SK,130
110 POKE (SK+56),130:REM  Relocate shift key
120 POKE CK,132
130 POKE (CK+56),132:REM  Restore control key
140 POKE 65378,0
```

10.7.3 Special Key Codes

The codes used by the special control keys are listed in 'Redefining the keyboard', in Part 4.

10.7.4 Function Keys

There are eight programmable function keys. They may be used as a quick method of entering frequently used commands or functions, or as eight separate interrupt keys.

Soft Keys

On entering the BASIC Interpreter, the bottom line of the LCD displays the default values. These may be changed at any time with the KEY function as described in Chapter 11. The screen only displays the first four characters, although the string may be up to fifteen characters long.

The display of the soft keys is entirely optional, but they are always active no matter whether they are displayed or not. This feature is controlled with the KEY ON and KEY OFF commands.

Interrupt Keys

The soft keys may be used to provide interrupts into BASIC programs. The relevant BASIC statements are:

```
ON KEY (n) GOSUB line
KEY (n) ON
KEY (n) OFF
KEY (n) STOP
```

The interrupts are tested at the beginning of every BASIC statement. Once a particular key has been pressed and has interrupted, then it cannot interrupt again, although other keys can still cause an immediate interrupt. However, one interrupt on a particular key can be held pending while its interrupt routine is executing. Function keys are recognised by having BIT 7 set (BIT 7 of the ASCII code). When in INPUT, an interrupt key if pressed will not return data to the variable, but will be remembered and acted upon after termination of the INPUT

10.8 Program Limits

10.8.1 Ranges

Variables	+9.99999E+126
String arrays	Up to 254 characters per string
Line numbers	0-65535 inclusive
Program line length	Up to 254 characters including line number

10.8.2 Precision

Single precision variables have 6 digit resolution. Double precision variables have 14 digit resolution.

10.9 Memory overhead

10.9.1 Program storage

Program lines require 4 bytes minimum, as follows:

Line number: 2 bytes
Line length: 1 byte
Carriage return: 1 byte

Also each reserved word, operator, variable name character, special character and constant character requires 1 byte. Maximum program size in present versions: 54K Bytes (Page 0)

10.9.2 Dynamic (Run-time) Memory Allocation

Symbol Table

Entries occupy 7 Bytes each

Maximum symbol table size: 16K Bytes

Single Precision Array

DIM A(X) occupies (X+1)*5+7 Bytes

Double Precision Array

DIM A!(X) occupies (X+1)*9+7 Bytes

String Array Elements

DIM A$(X,N) occupies (X+1)*N+7 Bytes

Array Size

The maximum number of elements in an array is 16,383.

10.10 Graphics

10.10.1 Introduction

Hunter 2 supports a graphics display of 240 x 64 pixels. Each point can be set to either white or black with the appropriate statement. Entering graphics mode is via the SCREEN statement, but this may be unnecessary since most of the functions which operate in graphics mode will automatically invoke the graphics screen.

The BASIC graphics keywords are:

CHAR	LOCATE
CIRCLE	POINT
CLS	PRESET
LINE	PSET

(Similar facilities are also available from assembly language programs: see Part 4.)

SCREEN 1 enters graphics mode, resets the character set to CHAR O and clears the display

The origin of the graphics screen is the top left hand corner, with the co-ordinates (0,0). So PSET(0,0) will set the corner pixel black. The origin for the LOCATE function, to determine the starting point for displaying ASCII characters, is (1,1) in the top left hand corner

Co-ordinates when specified are always absolute, ranging 0-239 for x and 0-63 for y (If two pairs of co-ordinates are required, then the first pair may be defaulted to the last plotted position.)

The above functions can be made to draw characters or lines in either white on black or black on white. This is achieved by appending another parameter after the command. For example, to print inverse characters of medium size, use:

```
CHAR 2,1
```

The '1' means inverse.

A detailed description of the graphics commands can be found in Chapter 11.

Graphics mode on the **Hunter 2** does not directly support the simple cursor manipulation codes: cursor up, cursor right, line feed, etc.

10.10.2 Graphics Demonstrations

The following program will draw a number of circles across the screen and back again:

```
10 SCREEN 1
20 FOR K=1 TO 167 STEP 4
30 CIRCLE (K+30,32),30
40 NEXT K
50 CLS
60 FOR K=167 TO 1 STEP -4
70 CIRCLE (K+30,32),15
80 NEXT K
90 CLS
100 GOTO20
```

The LINE command may also be used for drawing boxes as well as drawing lines. The following command will draw a line from the co-ordinates 100,40 to 200,20, the top left hand corner of the screen being 0,0:

```
10 LINE (100,40)-(200,20)
```

The following will draw a box with the bottom left hand corner at point 100,40 and the top right hand corner at 200,20:

```
10 LINE (100,40)-(200,20),1,B
```

if the second parameter is used then the first must also be specified. The box may be filled in by appending 'F' to the second parameter

The graphics cursor can be moved to any point on the screen by using the LOCATE command. Pixels are set by using PSET or cleared by PRESET The following example will produce a dotted line across the screen:

```
10 SCREEN 1
20 FOR X=0 TO 239 STEP 4
30 PSET (X,30)
40 NEXT X
50 END
```

10.11 Using Machine Code

Hunter BASIC is fast and flexible but may not always be fast enough or provide exactly the right facility to satisfy your needs. On these occasions, you can extend its facilities by writing a machine code routine (using an assembler package) and use that from BASIC.

This section describes how to use machine code routines from BASIC. It also provides general guidelines for controlling **Hunter 2** by using PEEK and POKE commands.

For details of writing assembly language programs, the assembly language environment, **Hunter 2** memory locations and ports, see Part 4.

The features described here should not be used by programmers who are unfamiliar with machine code programming.

10.11.1 The Microprocessor

Hunter 2 uses the CMOS NSC-800 microprocessor, which provides an instruction set largely compatible with the popular Z80 micro-processor Refer to any of the standard texts on Z80 machine code programming for more details.

It is not recommended, however, that the stack pointer register (SP) is used in a fashion incompatible with supporting interrupts. Standard use in the form of subroutine CALLs, PUSHs and POPs, however, is supported.

10.11.2 Calling a Machine Code Routine

To execute a machine code program from BASIC, use:

X=CALL(*addr*)

This passes control from BASIC to the address specified by *addr*. The machine code routine returns to BASIC by using a RET instruction.

Passing Simple Parameters

To pass a 16 bit integer to the routine, use the ARG function before using CALL. For example:

```
A = ARG(5)
X = CALL(addr)
```


This passes the argument (5 in this case) to the C and E registers. In this case, setting C to 5 and E to 0.

C is assigned the least significant byte of the argument, E the most significant byte. In general:

```
A = ARG(n1*256+n2)
```

will set C to *n2* and E to *n1*

These registers are used for compatibility with the CP/M style system call structure.

The machine code routine returns the value of the A register on exit to BASIC, as the value returned by the CALL function. For example:

```
1000 X = CALL(addr)
```

will set X equal to the value of the A register on exit from the machine code routine.

Passing Multiple Variables

To pass more values between the BASIC program and the machine code routine, you may use the machine code linkage stack.

This is accessed in BASIC by PUSH and POP In the routine:

```
1000 PUSH X,Y
1010 A = CALL(addr)
1020 Y = POP(1)
1030 X = POP(1)
```

PUSH places the integer values of X and Y on the stack for the machine code routine to access. On return from the call, POP removes integer values from the stack. (The argument specifies how far to move the pointer before removing the integer)

The machine code routine is passed a pointer to the linkage stack in the HL register pair This points to the bottom of the stack on which the variables are placed:

```
X msb
X lsb
Y msb
Y lsb   <------- (HL)
```

In other words HL points at the LSB of the most recently PUSHed value. To access other values, HL is incremented. These values may either be used by the called routine or modified and used by BASIC. The routine does not have to preserve the contents of HL on exit.

Passing Arrays and Strings

While you cannot pass an entire array or string variable direct to a machine code routine, you can get a pointer to an array by using VARPTR and pass that, as it is an integer See section 10.5 for details of the structures used to store arrays and strings.

System Calls

Operating system routines can be used by making CP/M-compatible system calls. These calls are described in Part 4. They are used by calling location 5 with the call number in the C register. NO guarantee is made of the contents of undefined registers on exit and in general all register contents will be destroyed.

10.11.3 Using RAM

By setting DEFSEG to the required page, you can place machine code routines in a number of different places in RAM. However, the only place in common RAM that will always be available (whatever DEFSEG is set to) is the serial communications buffer. Since the communications are active continuously on receive, take care not to let programs be corrupted.

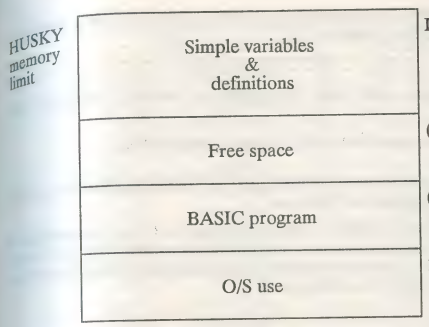
50 bytes are also available for small user programs, starting at the USER location (see Part 4).

Using the TPA

BASIC source in **Hunter 2** can extend for 54k, though there must be space for the symbol table.

The program and arrays are stored in lower memory upwards while simple variables and array definitions start at the top of the TPA (D700H) and work downwards. The gap in the middle may be used for machine code routines. The ALIM location contains a pointer to the start of the gap, while DEFLIM points to the end of the gap.

Figure 4.1: Storage Used by BASIC



Memory is allocated dynamically. Also, if CLEAR is used or a new program line is inserted, then the data storage is removed, so DEFLIM will point to the top of memory and ALIM to the top of the program. Hence, the program should be RUN before using these values. Clearly, if BASIC is only used to start a large machine code program then there will be no variables and only a small amount of BASIC source.

Using String Arrays

The best way to store machine code routines in BASIC is in a dummy string array, as described below:

a) First calculate the length of the machine code and dimension a string array with the required number of single byte elements using the formula `D=(N/2)-1` where N is the number of bytes needed for machine code and D is the number of single byte elements required in dimension.

   For example, to reserve 10 bytes of memory, the required dimension is DIM MC$(4,1).

   After dimensioning, all the bytes will be set to zero.

b) Find the start address of the memory reserved for the array. Assuming the array name is MC$, the statement:
   `AD=VARPTR(MC$)-1`
   will set AD to the address of the first reserved byte.

c) Put the machine code in the memory you have reserved, by using a series of POKE statements. For example:

```
POKE AD,30,7,14,1,205,5,0,201
```

NOTE: The machine code must be specified as decimal numbers in the POKE statement.

d) Finally, to execute the machine code, all that is required is:

```
X=CALL(AD)
```

where X is a dummy numeric variable and AD is the start address of the machine code as obtained in (b) above.

It is vitally important that DEFSEG is set to 0 when the CALL is executed. If DEFSEG is altered in the program, set it back to 0 before using CALL. It is also important that no attempt be made to use the array which is holding the machine code in the BASIC program as a string array

Using MAXFILES

BASIC reserves space in memory for file handling (buffer and file control block), which can be used for other purposes if not required for this. By default (on entry to BASIC), 168 bytes is reserved, for a single file. This can be changed by using MAXFILES:

```
MAXFILES=n
```

reserves space for *n* files. 168 bytes is reserved for each file, as follows:

n	START ADDRESS (HEXADECIMAL)	START ADDRESS (DECIMAL)
1	D657H	54871
2	D5AFH	54703
3	D507H	54535
4	D45FH	54367
5	D3B7H	54199
6	D30FH	54031
7	D267H	53863
8	D1BFH	53695
9	D117H	53527
10	D06FH	53359
11	CFC7H	53191
12	CF1FH	53023
13	CE77H	52855
14	CDCFH	52687
15	CD27H	52519

The end address of the area is always D6FFH (55039 decimal).

Code stored in this area is unaffected by the CLEAR command, but will be corrupted by using NEW (BASIC), FORMAT (DEMOS) and EDIT (BASIC or DEMOS).

If an application does not access files, it can use the whole space reserved using MAXFILES. If it needs to access files, it must only use those areas not used for files. BASIC allocates an area for each file that is opened, starting with the lowest area and working upwards.

For example, if an application set MAXFILES to 10, the area from D06FH to D6FFH would be set aside for files. If the application only ever had a maximum of three files open at a time, locations D267H to D6FFH could safely be used for other purposes.

Note that this method reserves space which is in the common RAM area, unlike using array space. Note also that the space allocated to file number one is always at the lowest point in memory and is not at a fixed address if DEFSEG varies.

10.11.4 Converting Machine Code to POKE Statements

When developing machine code for BASIC applications it is usually necessary to convert the machine code into BASIC POKE statements so that the application can load the machine code into memory prior to executing it.

The following BASIC program accepts a memory area start and end address, and also a start BASIC line number and then generates a file called POKE.HBA that contains a list of BASIC POKE statements which represent the area of memory given.

The file created, POKE.HBA, is in text form, not a tokenised form. This enables the file to be easily edited to contain the BASIC application, or transferred to a host system, and inserted using a word processor into a longer BASIC application program.

```
1 SCREEN1:CHAR0,1:PRINT"Machine code to poke statement program":CHAR0
5 INPUT"enter start location:",x
6 INPUT"last location :",z
7 INPUT"start line number :",y
8 CHAR0,1:PRINT"CONVERTING":CHAR0
10 OPEN"POKE.HBA"FOROUTPUTAS1
20 PRINT#1,STR$(y);"poke";STR$(x);
25 FORa=1TO16
30 PRINT#1,CHR$(44)+STR$(PEEK(x));
40 x=x+1:IFx=zTHENBEEP:PRINT"Completed! Program in POKE.HBA":END:ELSENEXT
50 PRINT#1,"":y=y+10:GOTO20
```


10.12 Data Storage Arrays

10.12.1 Types of Arrays

Captured data is generally held in array structures created within **Hunter 2**'s very large memory. There are three types of array:

- Simple Variables
- Double Precision
- Strings

10.12.2 Simple Variables

These are used for storage of decimal values of up to 6 digit accuracy.

10.12.3 Double Precision Arrays

These store numbers up to 14 digit accuracy, but use more memory.

Both these types store **much larger** numbers than indicated, but only by truncating the least significant digits. For example, the number 12345678 entered as a simple variable would be stored and reproduced as 12345600.

10.12.4 String Arrays

These store every character, whether numeric or otherwise, literally. They can have any number of characters (up to a maximum of 254 characters in each string). Facilities are provided for converting strings to numbers and vice versa.

10.12.5 Array Structures

Every array is denoted by a *name* and placed in memory in a sequential table. Multiple arrays are packed in memory by BASIC and accessed via a *Symbol Table*.

Individual array elements are identified by the array name and the element number, together with a character to identify the array type. Arrays start from element 0.

For example:

A(10) is the eleventh element of simple variable array A.

A!(2) is the third element of double precision array A! (quite different and independent from array A in the example above).

A$(0) is the first element in string array A$. A$(0) is identical with A$, which can be used for shorthand, useful if there are multiple references to a single string in the program.

In each case, the number in parentheses is the *element number*, and can also be:

a variable: A$(A)
or an expression: A$(A*5)
or another array element: A$(A(6))

Remember that element numbers start from 0!

10.12.6 Creating Arrays

Arrays can only be created once: if a subsequent attempt is made to re-define an existing array, an error occurs. This is to warn the programmer that he may be accidentally duplicating an array name.

If the statement DIM A$(1,6) is repeated **at any time** the following error message will appear:

```
*DIM Error
```

This can be easily avoided by incorporating the DIM in a program sequence that is only executed once, and subsequently branched around. This is often called an **Initialisation Sequence**. A typical format is:

```
10 IF Z<>0 THEN 100
20 DIM A$(1,6)
30 Z=55
```

The variable Z is a flag, indicating that the program has been run before. (All variables are cleared to 0 by CLEAR or any change to program content, but not by power off or removing batteries)

A slightly more sophisticated version is:

```
10 IF Z<>0 THEN 100
20 DIM A$(1,6)
100 Z=Z+1
```

Where Z acts both as a flag and as a counter of the number of times the program has been run, a very useful statistic!

10.12.7 Accessing Array Elements

Each element in a data array is accessible by its array index, expressed in parentheses. For example, A$(235) is the 236th element in A$.

Array indexes **must be** within the range declared in the corresponding DIM statement. **Hunter** BASIC warns the programmer of any attempt to access an element outside the declared range with:

```
*MAG Error
```

Be especially careful with incrementing indexes like FOR NEXT loops.

Array indexes often have a direct relationship to external variables, like stock codes. It is always preferable to access an array in this fashion in the interests of speed, rather than working sequentially through every element.

If there are discontinuities (gaps) in the existing codes, or if the numbers are too large, a conversion table or algorithm may be needed. Since a literal table (one entry for every code) is likely to be very inefficient, it is generally worth putting considerable effort into deriving an efficient 'tree' structure for direct array access.

10.12.8 Searching Arrays

There is often a need to search an array for an element of known content. In BASIC, this procedure can be painfully slow

One solution to this dilemma is to use a specialised machine-code subroutine to search the array and return with the index of the target element.

A much more practical method is the BASIC function SRCH.

10.12.9 Using Hunter 2's Screen

Hunter 2's large, 40 x 8 character, LCD screen can be used to positive effect to help its operator Because the screen is entirely flexible, these notes are provided as a source of ideas rather than as instructions.

This section refers to character-by-character use. More detailed information on direct dot addressing is available in section 10.10.

The Virtual Screen

Hunter 2's 8 x 40 character LCD screen acts as a window onto a much larger virtual (24 x 80) screen. When characters are written to the screen from a BASIC program they are always written to the virtual screen. The BASIC commands that move the window are CLS, INPUT, LOCATE and OPCHR 15.

CLS clears the virtual screen and sets the window to the top left of the virtual screen.

INPUT moves the LCD window to the part of the virtual screen containing the INPUT prompt.

LOCATE re-positions the cursor in the virtual screen and moves the window so that the cursor remains visible. To move the cursor without moving the window, use OPCHR 15.

Cursor Addressing

The cursor can be moved directly by a BASIC program, to allow you to write characters to anywhere on the virtual screen. Cursor addressing is relative to the virtual screen.

There are two ways of moving the cursor·

```
LOCATE X,Y
```
and
```
OPCHR 15,X,Y
```

where X and Y are the coordinates.

LOCATE moves the window to keep the cursor visible in the LCD screen. OPCHR 15 does not move the window.

Fast Screen Updating

You can use the virtual screen and cursor addressing to create up to six separate text screen displays and then switch between them rapidly, using a single BASIC instruction.

To create the text in a window, use OPCHR15,x,y to move the cursor and PRINT the text.

To display a window, use LOCATE x,y to move the cursor to the window. This will display that part of the virtual screen.

Should you not require to display the cursor, it can be switched off using the CUROFF command.

This simple program (FLIPSCRN.HBA) demonstrates this facility:

```
10 CLEAR: CUROFF: KEYOFF: CLS
20 GOSUB 1000
30 INKEY A: IF A=0 THEN 30 ELSE IF A<49 OR A>54 THEN BEEP: GOTO30
40 ON (A-48) GOTO 100,110,120,130,140,150
50 REM flip to a part of the screen
90 LOCATE 1,7: GOTO30
100 LOCATE 79,7: GOTO30
110 LOCATE 1,15: GOTO30
120 LOCATE 79,15: GOTO30
130 LOCATE 1,23: GOTO30
140 LOCATE 79,23: GOTO30
150 REM put the text in the virtual screen
999 FOR I=1 TO 6: OPCHR 15,1,I
1000 PRINT STRING$(38,65): NEXT
1010 PRINT STRING$(38,66): NEXT
1020 FOR I=1 TO 6: OPCHR 15,41,I
1030 PRINT STRING$(38,66): NEXT
1040 FOR I=9 TO 14: OPCHR 15,1,I
1050 PRINT STRING$(38,67): NEXT
1060 FOR I=9 TO 14: OPCHR 15,41,I
1070 PRINT STRING$(38,68): NEXT
1080 FOR I=17 TO 22: OPCHR 15,1,I
1090 PRINT STRING$(38,69): NEXT
1100 FOR I=17 TO 22: OPCHR 15,41,I
1110 PRINT STRING$(38,70): NEXT
1120 RETURN
```

10.12.10 Help Statement

The HELP facility provides a means of breaking into normal programs to display text for operator assistance.

After the HELP text has been read control is returned to the main program, as if nothing had happened, by pressing [HELP] again.

The Help Vector

The text displayed by HELP can be linked to the operation currently being performed. The start position of the text can be set under program control. This is achieved by the pair of RAM locations 'VECTOR', indicating the memory address of the start of text. This is controlled by the HELP function in BASIC.

VECTOR can also be set by POKE.

Text Storage

Help text is stored by means of REM statements, which allow storing of ASCII text within programs.

The address in the Help vector points to the start of the line number The Help program will skip forward over the first four bytes, checking the REM token and then displaying the text.

The REM token is stored as 143 Decimal, (8FH).

This line of BASIC source code:

```
10 REM ABCDEFGHI
```

Would be stored like this:

0E	0A 00	8F	41 42 43 44 45 46 47 48 49	0D
line length	line number	token	ASCII text	CR (line terminator)

HELP Text Display

The software which controls HELP will scroll forward through the lines of BASIC text, or pseudo-BASIC text, until the keyword found is not a REM. Scrolling backwards is also allowed until a line is found without a REM. In each case, further scrolling is not allowed.

Storing the Current Display

The contents and cursor position of the screen are not lost while in HELP mode. On exit from HELP mode, the screen and cursor are restored to their states prior to HELP mode entry

Action during HELP

When a program is interrupted for HELP display all normal action is suspended. Serial data reception will continue up to the capacity of the receive buffer. If handshaking is enabled then it will continue in a transparent fashion.

Programs will continue after [HELP] is pressed again.

HELP will be entered whenever the user presses [HELP]. The program does not have to be inputting text from the keyboard.

Graphics Mode

Because **Hunter 2**'s graphics screen cannot be re-created from RAM memory, HELP text would destroy the contents of the screen if allowed during graphics mode. For this reason, HELP is inhibited in graphics mode and [HELP] depressions are ignored.

10.13 Increasing Battery Life

Hunter 2's power usage depends on which operating mode it is in:

- TURNED OFF· minimal current
- POWER SAVE (awaiting keyboard or serial port entry): 15mA
- OPERATING (running a program for calculation, data storage, screen updating or any operation involving continuous execution of user program): 80-90mA.
- COMMUNICATING (connected to a fully equipped RS232 port and operating): 115-125mA.

To maximise battery life, you should attempt to maximise the time spent in POWER SAVE mode. In a BASIC program, **Hunter 2** enters this mode when you execute an INPUT or INCHR command, but NOT when you use the INKEY$ function to scan the keyboard. Therefore, you should not design programs that wait in a loop using INKEY$ to scan the keyboard! If this type of facility is required use the following routine instead of INKEY$:

```
100 REM Power save keyboard scan
110 Z=ARG(47)
120 A=CALL(5)
130 A$=CHR$(A)
140 RETURN
```

(This uses system call 47 to scan the keyboard. See Part 4 for more details.)

If an application uses communications, the RS232 port turns on automatically. Once communication has finished, it should turn the interface off by using the command OUT 132,0.

10.14 Low Power Warning

When **Hunter 2**'s batteries become exhausted a power warning message will appear on the screen. Normal operation will continue, but keyboard entries will be punctuated by warning tones and repetition of the power warning message until the battery state is rectified either by replacement of primary cells or recharging of Nickel-Cadmium cells, if installed.

Eventually, if the warnings are ignored, **Hunter 2** will shut down and refuse to operate further

Every **Hunter 2** has to pass a stringent operating test in this low power regime. However, it is strongly recommended that operator training procedures and user programs are structured to avoid continued operation when warnings become persistent, especially when rechargeable cells are used.

The power warning sequence may be misleading if RS232 communications are used infrequently. When **Hunter 2**'s communications system is activated, the power drain increases sharply. This increase can cause the power warning stage to be missed altogether, causing **Hunter 2** to shut down without warning.

While this is not hazardous, it can be very confusing for an operator who does not know what is wrong. Because of this it is recommended that user programs with infrequent communication requirements use the following test routine. This routine activates the communications package, and then tests the power state before restoring **Hunter 2** to normal. This routine is used at the start of every data entry sequence.

If batteries become exhausted whilst communications are in progress, then normal power warning messages will appear once keyboard operation is resumed.

```
10 REM example warning routine
20 OUT 132,1:REM power up comms port
30 J=INP(2):J=J AND 4:K=INP(2):K=K AND 4
35 OUT 132,0
40 IF (J AND K) = 0 THEN 80
50 CLS
60 PRINT "PLEASE CHANGE MY BATTERIES!"
70 BEEP:BEEP:BEEP:BEEP
80 RETURN
```

This routine operates as follows:

* The power status is available as bit 2 in **Hunter 2** port 2.
  0 = power OK, 1 = Low Power state.
* Line 20 turns on the RS232 interface.

* Line 30 reads the power status from port 2 twice, to allow the program to ignore momentary power fluctuations.
* Line 35 turns off the RS232 interface, to conserve power.
* Line 40 checks the power status readings.
* Lines 50-80 display a warning message and make a warning sound, if power is low

10.15 Power Handling Commands

10.15.1 Introduction

BASIC supports several powerful functions associated with [PWR]. BASIC programs can control precisely what happens when **Hunter 2** is turned on, or even not allow users to turn **Hunter 2** off. These facilities are summarised here for your convenience.

10.15.2 Default Conditions

[PWR] behaves as normal - at any stage it can be used to turn **Hunter 2** on or off. The current execution position is lost on turning the power off, and turning it back on returns to the File Manager However the current BASIC program and variables are retained.

10.15.3 POWER CONT

* [PWR] is disabled.
* The automatic timeout power down (if no key is pressed within a set period of time) is disabled.

10.15.4 POWER n

* This command overrides a previous POWER CONT
* n refers to the automatic power down timeout:
    * n must be in the range 10-255 or 0
    * n = 0 is no automatic timeout
    * n is in multiples of 5 seconds

10.15.5 POWER OFF

* Turns the power off.
* When power is turned on, BASIC program execution starts at the beginning.

10.15.6 POWER OFF, RESUME

As POWER OFF, but turning **Hunter 2** on continues execution where it left off.

10.15.7 ON POWER GOTO

When [PWR] is pressed (during program execution) program execution goes to the specified line number

10.15.8 ON POWER RESUME

* Powering down proceeds as normal.
* Powering up begins program execution where it left off.

10.15.9 Timeouts

Hunter 2 has an automatic power-down timeout, which can be modified using POWER n. If **Hunter 2** has been switched off, this timeout is reset to the default 5 minutes unless CONT (see Part 2) or POWER OFF, RESUME (see section 10.15.6) has been used.

10.16 File Handling

10.16.1 Introduction

BASIC provides the following keywords for file handling:

CLOSE	MAXFILES
EOF	NAME
INPUT#	OPEN
KILL	PRINT#
LOC(n)	WRITE#

These provide direct support for sequential access to files stored in **Hunter** 2's RAM disk.

By using machine code facilities, random access is also possible.

10.16.2 File Numbers

When files are OPENed for input or output they are assigned a 'file number' which is specified by the AS parameter The 'file number' cannot be greater than the value specified in MAXFILES. If no MAXFILES have been specified a default value of one is taken. A file is always referred to by its 'file number'

10.16.3 File Handling Commands

A brief description of the file handling commands and statements follow, further details may be found in the relevant sections of Chapter 11.

CLOSE#

When files have been OPENed for reading or writing and no further access is required CLOSE will write the end of file marker (EOF) to the file and ensure that its file control block (FCB) is written back to the directory.

EOF

When reading data from a file a check must always be made after every INPUT# statement for end of file, an attempt to read past the end of file will produce an error

INPUT#

This command is used for reading data from a file.

KILL

Used for deleting files from within a BASIC program. The file must first be CLOSEd if it has previously been OPENed for reading or writing.

LOC(n)

This will return the number of records that have been read or written to a file. The parameter (n) refers to the 'file number' associated with the file.

MAXFILES

Sets the maximum number of files that may be opened simultaneously; if it is not specified a default value of one is taken. MAXFILES will perform a CLEAR operation when it is executed, destroying all symbol table entries. Therefore it must be defined at an early point in the program.

NAME

This will re-name a file from within a BASIC program.

OPEN

Before a file can be read or written to it must first be OPENed. The OPEN command has as one of its parameters FOR, which has to be specified with one of three options which are:

OUTPUT	Opens and clears file ready for output.
APPEND	Opens a file for output. Records written are added to the end of the file.
INPUT	Opens a file for reading.

PRINT#

Data is output to the file. Commas or CRs (CHR$(13)) should be written between each data item to simplify reading the data back, or each item should be written by a separate PRINT# instruction.

WRITE#

Data is output to the file. WRITE# writes commas between each data item automatically, so the instruction does not need to write commas itself.

10.16.4 Determining Disk Space

BASIC does not provide a function to determine free disk space directly, but you can deduce it indirectly as follows.

The RAM disk is divided into 2K blocks of memory termed allocation units. The operating system records which blocks are free by setting bits in a bit map held in memory. If a bit is set, the corresponding block is allocated (used); if it is clear (0), the block is unallocated (free)

The operating bit map is in page 1 of the RAM disk area, but it is easier to access the copy held in common RAM area between 55168 and 55232.

The following program shows how to calculate the free space by counting the number of free blocks indicated by the bit map:

```
5 REM Free disk space calculation
10 bm=55168: REM start of bit map
20 t=0
30 FORj=bmTObm+63
40 s=8
50 n=PEEK(j)
60 IFn=0THEN90
70 FORi=0TO7:n=2*n:IFn>255THENn=n-256:s=s-1
80 NEXTi
90 t=t+s
100 NEXTj
110 PRINT"Disk space free is";2*t;"K bytes"
```


10.17 Random Access Filing

Hunter 2's operating system provides the CP/M-compatible READ and WRITE RANDOM calls. These can be used for random access to **Hunter 2** files, from within BASIC. This is particularly useful for applications such as stock control or inventory programs, where sections of files need to be inspected or modified.

When a file is opened in BASIC, an entry is made in the symbol tables which describes:

- where the file's FCB is located
- where the file's DMA is stored
- whether the file was opened for output, input or append. The FCB is automatically initialised by the Interpreter and the file is opened or made depending on the command entered.

If several files are created, then each file will have a symbol table entry and its own FCB and DMA buffers.

The BASIC Interpreter imposes stringent controls on the file commands entered in order to protect the user's files. For example, if a write is attempted to a file which has been opened for input, an error will occur, rather than corrupt the file.

These safeguards are not provided when access to file is made directly via machine code calls, so extra care must be taken when designing the machine code.

There are six stages required in implementing random calls within BASIC. These will be described in detail, together with an example program.

10.17.1 Stage 1 - Installing Machine Code

The following commands can be used to create the code from BASIC:

```
POKE 52984, 14, 22, 24, 14, 14, 40, 24, 10, 14, 16, 24, 6, 14, 33, 24, 2
POKE 63000, 14, 34, 197, 14, 26, 17, 128, 0, 205, 5, 0, 193, 17, 7, 213, 195, 5, 0
```

The machine code created is as follows. This must be loaded into a common area of RAM from which execution will occur.

A suitable area is the user RAM area USER (see Part 4), where the code will remain permanently unless modified by the user

Placing the machine code in RAM allows you to modify it easily (for different FCBs and DMAs) by poking the relevant locations, reducing the amount of machine code required.

```
F608 0E16     MAKE:     LD C,16H      ;[62984] MAKE
F60A 180E               JR DOIT
F60C 0E28     FILL:     LD C,28H      ;[62988] WRITE BLOCK FILL
F60E 180A               JR DOIT
F610 0E10     CLOSE:    LD C,10H      ;[62992] CLOSE
F612 1806               JR DOIT
F614 0E21     READR:    LD C,21H      ;[62996] RANDOM READ
F616 1802               JR DOIT
F618 0E22     WRITER:   LD C,22H      ;[63000] RANDOM WRITE
F61A C5       DOIT:     PUSH BC
F61B 0E1A               LD   C,1AH    ;Set DMA address
F61D 18000    LD   DE,0080H           ;E(L)=63006, E(H)=63007
F620 CD0500   CALL 5H
F623 C1                 POP  BC                   ;Load FCB address
F624 1107D5   LD   DE,0D507H          ;E(L)=63013, E(H)=63014
F627 C30500   JP   5H
              ;
              ;
F62A          END
```

10.17.2 Stage 2 - File Initialisation

The number of files which can be opened simultaneously in BASIC must be defined at the start of the program. Although there is a default in BASIC of one file, to simplify the applications description a MAXFILES statement MUST be used even if this is of the form MAXFILES = 1.

Consider MAXFILES = 3

The BASIC Interpreter creates the space for the FCBs and DMAs for each of these files. Additional information relevant only to BASIC is also created for each file, resulting in 168 bytes being allocated in total for each file.

It is necessary now to 'MAKE' the three files to be used for the application. In order to keep the machine code required to a minimum, you can use BASIC to set up the FCBs with the filename prior to executing the call to make the file in machine code.

At this stage it is worth commenting that BASIC does not create a file when you OPEN it for OUTPUT If this was done, a file 2K in length would be created, as BASIC needs to insert a terminator character (1AH) in the file when closed, resulting in a file of minimum length 2K bytes being created.

Consider the creation of three files, TEST1.DTA, TEST2.DTA and TEST3.DTA. The three files must not already exist. If they do, they must be KILLed by BASIC.


```
10 ON ERROR GOTO 10000
20 OPEN"TEST1.DTA"FOR INPUT AS1
30 OPEN"TEST2.DTA"FOR INPUT AS1
40 OPEN"TEST3.DTA"FOR INPUT AS1

10000 CLOSE
10010 A=CALL(62984):A=CALL(62992):RESUME NEXT
```

This code traps an error as soon as line 20 is executed, as TEST1.DTA does not as yet exist.

Execution then goes to line 10000 where the file is CLOSEd by BASIC, and then machine code is executed to first MAKE and then close the file. This creates a zero length file called TEST1.DTA.

NOTE: In BASIC as the file was OPENed for INPUT, the action of the BASIC statement CLOSE is to inform the Interpreter that the file is no longer available for use, the BASIC CLOSE statement does not close the file in this case as the file was read only

Observe that the file has been opened as #1, as the relevant FCB for this file is already loaded into the machine code.

The Interpreter is then told to RESUME NEXT, which causes the Interpreter to continue execution following the statement in which the error occurred, in this case line 30.

The procedure is repeated until three zero length files have been created which concludes the file initialisation stage.

This stage should only be executed once for the files, so the BASIC application program should be written to ensure this is the case.

10.17.3 Stage 3 - Opening the Files for Use

By this stage the files should have been created as zero length files in the disk space.

To initialise the FCBs for the machine code, the files are opened by lines 50 to 70. As inputting from a zero length file causes an error, line 45 traps errors, diverting them to the error handler at line 65000 which simply ignores the error.

```
45 ON ERROR GOTO 65000
50 OPEN"TEST1.DTA"FOR INPUT AS #1
60 OPEN"TEST2.DTA"FOR INPUT AS #2
70 OPEN"TEST3.DTA"FOR INPUT AS #3
75 ON ERROR GOTO 0
65000 RESUME NEXT
```

Once the files are opened, it is advisable to cancel the error trapping so that TRUE errors are shown.

The organisation of the FCBs and the addresses which are relevant to this particular application is as follows:

File number 3: fcb = D657H...D67AH
r0 = 54904D (D678H)
r1 = 54905D (D679H)
r2 = 54906D (D67AH)
dma = D67FH

File number 2: fcb = D5AFH...D5D2H
r0 = 54736D (D5D0H)
r1 = 54737D (D5D1H)
r2 = 54738D (D5D2H)
dma = D5D7H

File number 1. fcb = D507H..D52AH
r0 = 54568D (D528H)
r1 = 54569D (D529H)
r2 = 54570D (D52AH)
dma = D52FH

(Non-essential addresses have been omitted for clarity)

NOTE: The order in which files are OPENed in BASIC will alter where each file's FCBs are located in memory Care should be taken to adhere to this example or verify that the locations are correct for your application.

10.17.4 Stage 4 - Transferring Data to and from Files

Transferring data to and from the files can be performed in several ways. The direct way is to poke the information into the relevant DMA buffer for the file concerned and then executing the WRITE call. Data can be read from the file by performing the READ and then PEEKing the information from the DMA buffer This method can be used, but is slow and laborious.

A subtler method can be employed with a little knowledge of BASIC's array and string storage methods, giving a more convenient direct interface between BASIC and the file's DMA buffer

If string information is to be handled, the DMA buffer can be a string. The string must have a length of at least 128 bytes. For convenience in this application, the length byte has been incorporated into the DMA buffer which simplifies the string manipulations, but which means that the actual sector storage area is reduced by one byte to 127 bytes.

When the full 128 bytes are required, the user must ensure that the string length is always made up to 128 bytes. By using the VARPTR statement, the address of the string can be found and substituted into the DMA address of the machine code prior to a read or write to the file.

Consider·

```
DIM D$(0,128)       :REM dimension string
W=VARPTR(D$)-1      :REM point to length byte
H=FIX(W/256):L=W-256*H
```

where H = high order address byte and L = low order address byte of the DMA buffer string D$.

If numeric data is to be used, the DMA buffer can be a single precision array. The number of elements in the array must be not less than 25 in order for the array to be not less than 128 bytes in length.

WARNING· If the array occupies less than 128 bytes, arrays following the array used as the DMA buffer will be corrupted. The DMA buffer is always 128 bytes in length.

Consider·

```
DIM D(25): REM dimension string
W = VARPTR(D(0))-4
H = FIX(W/256):L=W-256*H
```

where H = high order address byte and L = low order address byte of the DMA buffer array D(x)

Once the DMA method has been chosen, it is then simply a matter of:

i) setting the DMA address to be used

ii) setting the relevant FCB address

iii) loading the random sector number bytes and executing the random READ or WRITE call.

For example, to write to sector X.

```
POKE 54904,X
POKE 63006,L,H
POKE 63013,87,214
A = CALL(62988)
```

And to read from sector X.

```
POKE 54904,X
POKE 63006,L,H
POKE 63013,87,214
A = CALL(62996)
```

NOTE: In either case, if A is non-zero, the read or write has failed either due to the FCB information being incorrect or, in the case of a write, there being insufficient space on the disk.

10.17.5 Stage 5 - Closing Files in Machine Code and BASIC

Care must be taken in designing the application to ensure that the files are closed after their use. Failure to close the files will result in loss of the data from the files.

The file must be closed in machine code first and then with the BASIC statement CLOSE in order to indicate to the Interpreter that the files are closed.

In order to close the file in machine code only, the FCB address is required for the file concerned:

```
POKE 63013,87,24
```

and the relevant machine code call must be made to CLOSE a file:

```
A = CALL(62992)
```

Once all files have been closed in machine code, a general CLOSE statement should be made in the BASIC program.

10.17.6 Demonstration Program

The following simple program demonstrates the principles outlined previously. It is a simple inventory style program which stores a customer name, address and telephone number, a stock number, a minimum stock level value and then maintains a stock count for that customer which can be increased or decreased.

A customer is accessed through customer number which is, in fact, the sector number of the entry in the files relating to that customer

The program has been kept simple but functional in order that the principles to be demonstrated are not clouded by ancillary software.

Both string and array 'DMA' methods are used to give the user some experience of the techniques; three files opened simultaneously and accessed randomly.

The demonstration program has been organised as a number of separate stages, which correspond to the various stages described earlier:

Stage 1.	lines 20,30	Loading machine code
Stage 2:	lines 40-94	File Initialisation
Stage 3:	lines 100-120	Opening files
Stage 4:	lines 160-180 and 50040-50050	Transferring data
Stage 5:	line 60000	Closing files

In the next example, three files are used: two to store two string DMA buffers S$ and T$, and the third to store an array DMA buffer D(0).

S$ stores the name and address string
T$ stores the town and telephone number string
D(0) = 55 if the entry exists, else D(0) = 0
D(1) = stock number
D(2) = minimum stock level
D(3) = current stock level

D(3) is kept greater or equal to zero.

Observe in lines 50040-50050 that the machine code call to WRITE WITH BUFFER FILL is used. This performs exactly as the normal random WRITE call, but has the advantage that if the block does not exist it clears the entire block before performing the actual sector write. This has a particular advantage when using 'array DMA' buffers as if a read is performed on a new section, the array is initialised to zero for the user

Although not included in the demonstration package, it would be advantageous to check the value of 'A' for non-zero in lines 50040-50050 after the call to WRITE in order to detect a full disk or failed write for some other reason, and flag this to the user.

```
5 REM **  RANDOM FILE DEMONSTRATION  **  (J.J.B)
10 ONPOWERGOTO60000:IFwz=55THEN100  'trap power key and test if initialised
REM Set up the machine code at F608H
20 POKE62984,14,22,24,14,14,40,24,10,14,16,24,6,14,33,24,2
30 POKE63000,14,34,197,14,26,17,188,0,205,5,0,193,17,7,213,195,5,0

REM Initialise the dma buffers to be used
40 MAXFILES=3:DIMt$(0,130 ,s$(0,130),d(26)
50 w=VARPTR(d(0))-4:hd=FIX(w/256):ld=w-256*hd
60 w=VARPTR(s$)-1:hs=FIX(w/256):ls=w-256*hs
70 w=VARPTR(t$)-1:ht=FIX(w/256):lt=w-256*ht

REM Make and close the file in machine code
80 ONERRORGOTO20000
90 OPEN"test1.dta"FOROUTPUTAS1
92 OPEN"test2.dta"FOROUTPUTAS1
94 OPEN"test3.dta"FOROUTPUTAS1
96 wz=55

REM Use basic to open the files initially
98 ON ERROR GOTO 65000
100 OPEN"test1.dta"FORINPUTAS1
110 OPEN"test2.dta"FORINPUTAS2
120 OPEN"test3.dta"FORINPUTAS3
125 ON ERROR GOTO 0

REM Main program loop starts here
130 POKE54905,0,0:POKE54569,0,0:POKE54737,0,0   'initialize the r1, r2 of fcb's
140 CUROFF:SCREEN1:CHAR2,1:PRINT "Customer record";
150 CHAR0:LOCATE40,35:INPUT"Enter customer number >",X:SCREEN0

REM Read the sector X of each file
160 POKE54736,X:POKE63006,ls,hs:POKE63013,175,213:a=CALL(62996)
170 POKE54904,X:POKE63006,ld,hd:POKE63013,87,214:a=CALL(62996)
180 POKE54568,X:POKE63006,lt,ht:POKE63013,7,213:a=CALL(62996)

REM If failed to read then assume it to be a new sector
190 IFa<>00Rd(0)=0THENGOSUB50000
200 GOSUB1000:GOTO130

REM Display routine to take file data and display it to the operator
1000 KEYOFF:CPCHR1:PRINT"Cust No # ";X
1010 LOCATE2,2:PRINTLEFT$(s$,20);:LOCATE2,3:PRINTMID$(s$,21,30);
1020 LOCATE2,4:PRINTLEFT$(t$,20);:LOCATE2,5:PRINTMID$(t$,21,20);
1030 LOCATE20,2:PRINT"stock No ";d(1):LOCATE20,3:PRINT"min stock";d(2)
1035 LOCATE20,4:PRINT"stock lev ";d(3)

REM Section to ask for the stock level update
1040 BEEP:LOCATE5,7:INPUTUSING("NNNN",0,4);"Enter stock change  :",y
1050 d(3)=d(3)+y:IFSGN(d(3))=-1THENd(3)=0
1060 GOSUB50040:RETURN

REM Subroutine to close the file in basic, then Make and Close the file
20000 CLOSE:a=CALL(62984):a=CALL(62992):RESUMENEXT

REM Subroutine to CLOSE files in machine code
40000 POKE63013,7,213:a=CALL(62992)
40010 POKE63013,175,213:a=CALL(62992)
40020 POKE63013,87,214:a=CALL(62922):RETURN
```



```
REM Routine to obtain new data from the user
50000 CLS:INPUT"Enter name:  ",a$:INPUT"   address: ",b$
50020 INPUT"      town: ",c$:INPUT"    tel no: ",d$
50025 INPUTUSING("NNNN",0,4)"Enter stock no :",d(1)
50026 INPUTUSING("NNNN",0,4)"Enter min stock level :",d(2):d(0)=55:d(3)=0

REM Arrange data in convenient way into the 'DMA buffer strings'
50030 s$=a$+SPACE$(20-LEN(a$))+b$
50035 t$=c$+SPACE$(20-LEN(c$))+d$

REM Subroutine to write sector X to the files
50040 POKE54568,X:POKE63006,lt,ht:POKE63013,7,213:a=CALL(62988)
50045 POKE54736,X:POKE63006,ls,hs:POKE63013,175,213:a=CALL(62988)
50050 POKE54904,X:POKE63006,ld,hd:POKE63013,87,214:a=CALL(62988):RETURN

REM Routine to trap user turning off machine so as to close files correctly
60000 GOSUB40000:CLOSE:POWEROFF
65000 RESUME NEXT
```

10.18 Executing Programs from the RAM Disk

10.18.1 Executing a Single Program

BASIC programs are usually run by loading them into BASIC's workspace. However, the operating system also provides a unique ability to run programs in situ in the RAM disk.

This facility is provided by the File Manager command /DO. It leaves the entire TPA free for storage or variables and arrays, to maximise performance. The program file must be prepared in a special way to use this facility. For further details, see /DO in Part 2.

10.18.2 Program Switching between the TPA and the RAM Disk

It is also possible to jump between programs resident in either the TPA or the disk space, simulating a GOTO filename command. This allows you to create BASIC programs that are much bigger than the limiting TPA size, while avoiding the speed limitation entailed by more conventional CHAINing of programs.

To switch back and forth between a program in the TPA (PROGRAM1) and another program within the RAM disk (PROGRAM2), you must first store the disk program as described under the /DO command in Part 2 and then place the machine code routine in common RAM (F605H in this example) by using the following statements:

For 9G08, use:

```
POKE 62981,61,129,33,1,8,24,5,62,128,33,1,1,50,241,252,50,146,249,34
POKE 63000,11,248,205,159,228,195,16,14
```

For 9G06, use:

```
POKE 62981,61,129,33,1,8,24,5,62,128,33,1,1,50,241,252,50,146,249,34
POKE 63000,11,248,205,169,228,195,16,14
```

For 9G, use:

```
POKE 62981,61,129,33,1,8,24,5,62,128,33,1,1,50,241,252,50,146,249,34
POKE 63000,11,248,205,169,228,195,56,14
```

To switch from PROGRAM2 to PROGRAM1, use the following statement:

```
A=CALL(62988)
```


To switch from PROGRAM1 to PROGRAM2, use the following statement:

```
A=CALL(62981)
```

Variables are shared between the two programs automatically

When execution switches, it begins at the start of the other program. To avoid this restriction, include statements at the start of the program to branch to the appropriate section of the program depending on a supplied parameter, eg by using GOTO *variable*.

The machine code routine switches the operating system's page pointers. The source is included here for interest:

```
;       This link program allows the execution of a program
;       resident in the file space and execution of a program
;       in the TPA

;This link program is dependant on the version of operating system:

        ASEG    LINK
        ORG     0F605H

BSPAGE  EQU     9G          0FCF1H; 9G06h -     ; 9G08h -
PAGFLAG EQU                 0F992H;       -     ;       -
BOFA    EQU                 0F80BH;       -     ;       -
ZRUN    EQU                 0E35H;        0E0Dh ;       0E0Dh
CLPAGE  EQU                 0E4A9H;       -     ;       0E49Fh

;       Select to execute program from page 1 disk area

LINK1:  LD      A,81H                           ;A = CALL (62981)
        LD      HL,00801H
        JR      LINKX

;       Select to execute program from the TPA

LINK0:  LD      A,80H                           ;A = CALL (62988)
        LD      HL,0101H

LINKX:  LD      (BSPAGE),A
        LD      (PAGFLAG),A
        LD      (BOFA),HL
        CALL    CLPAAGE
        JP      ZRUN+3

        END
```

1) The file space program:

NOTE: This program must be the first program in the file space:

```
10  FOR P = 1 TO  10
20  I = I + 1
30  PRINT "File  ";I
40  NEXT
50  A = CALL (62988)
```

Explanation:

Lines 10-40 print and increment the common variable I
Line 50 executes a call to the GOTO program in the TPA

2) The TPA program (for 9G08h):

```
10 POKE 62981,62,129,33,1,8,24,5,62,128,33,1,1,50,241,252,50,146,249,34
20 POKE 63000,11,248,205,159,228,195,16,14
30 FOR P = 1 TO  10
40 I = I  + 1
50 PRINT "TPA ";I
60 NEXT
70 A = CALL (62981)
```

Explanation:

Lines 10-20 load machine code into common ram
Lines 30-60 print and increment the common variable I
Line 70 executes a call to the GOTO program in the file space

10.18.3 Program Switching Multiple Programs in situ within the RAM Disk

The facility described in section 10.18.2 can be extended to multiple files by preparing the files and using a machine code routine as described here. This also allows the programs to share variables and arrays. (To avoid sharing variables, execute a CLEAR statement on entry to each program.)

Technique

The following example shows how to load three files into a 208K machine and then use them. You may load any number of files in this way, provided this procedure is followed exactly

Before you begin, make sure that the BASIC programs are backed up to another device, as any errors in procedure may result in losing files.

To execute from the RAM disk, a program must be loaded into the start of a RAM disk page. To load a single file in this way, you can simply format the disk and then load the file. To load multiple files, you must load dummy files between program files, to force each one to load at the start of a page.

A **Hunter** 2 disk page consists of 48K bytes of memory. The minimum block of memory that may be allocated to a file is 2K bytes. A 2K block of memory in each page used for the RAM disk is used for the directory, so the largest file that can be stored contiguously in a single disk page is 46K.

If, for example, a **Hunter** BASIC program occupies 10K of disk space, to ensure that a second program is loaded from the start of the next disk page, a dummy file must be saved first of 36K. Once the second BASIC program is loaded, this dummy file can be deleted if the disk space is required.

The following procedure outlines how to load the demonstration programs listed at the end of this section, into a 208K **Hunter 2**.

The application consists of three simple BASIC programs PROG1.HBA, PROG2.HBA and PROG3.HBA. Each file occupies 2K of disk space.

PROG1.HBA initialises the machine code to perform the linking to the other program and then each program creates its own string array and displays the information. On pressing ENTER, the program calls the machine code which causes the program execution to move to the next page in which the next program resides. Each program has its own variables and may have access to the variables of the other program if necessary The TPA is used exclusively for the storage of arrays and variables for all the programs.

PROG1.HBA places the programs into CONT mode, making the emergency escape sequence necessary if it is required to escape from the program.

- Format **Hunter 2**
- In BASIC LLOAD the file PROG1.HBA and SAVE "PROG1.HBA" or load a tokenised BASIC program using INP from DEMOS.
- From DEMOS observe that the program loaded occupies 2K of disk space. PROG1.HBA will be loaded starting at the bottom of Page 1 disk space.
- To load the next program file at the start of Page 2 of disk space, 46K - 2K of space must be occupied by a dummy file, so in DEMOS, type the command:

  `SAVE 176 DUMMY1.COM` ENTER

  SAVE blocks are 256 bytes, so 44K is 4*44 ie 176 blocks.

Use STAT to check that:
PROG1.HBA is 2K
DUMMY1.COM is 44K

* Type NEW in BASIC and LLOAD the file PROG2.HBA, and SAVE "PROG2.HBA"

* In DEMOS observe that PROG2.HBA occupies 2K of memory. PROG2.HBA will be loaded starting at the bottom of Page 2 disk space.

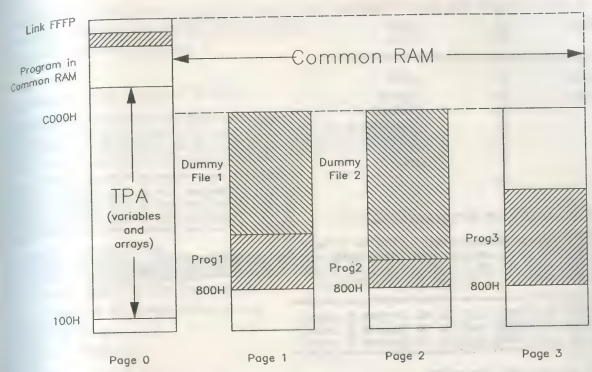
* To load the next program at the start of Page 3, 46K - 2K of space must be occupied by a second dummy file, so in DEMOS type the command:

```
SAVE 176 DUMMY2.COM (ENTER)
```

Use STAT to check that:
PROG1.HBA is 2K
DUMMY1.COM is 44K
PROG2.HBA is 2K
DUMMY2.HBA is 44K

* Type NEW in BASIC, LLOAD the file PROG3.HBA and SAVE it as PROG3.HBA

At this stage all three files will be resident in the disk space, like this:



* From DEMOS type the command:

```
/DO PROG1.HBA (ENTER)
```

This runs the programs, causing each program to link to the next on operation of (ENTER).

Execution of the call to link to the next program causes execution to commence from the start of the program. By using variables as flags (A(1), A(2) and A(3) in the example programs) it is possible to cause program execution to branch to different areas if required.

```
;=============================================
; IN-SITU execution of multiple files resident in file space
;=============================================

; the link program is dependant on the version of operating system as follows
bapage:     equ         9G  0fcf1h ;9G06h -     ;9G08h -
pagflag:    equ          0f992h ;     -     ;     -
bofa:       equ          0f80bh ;     -     ;     -
zruns:      equ          0e35h  ;     0e0dh ;     0e0dh
clpage:     equ          0e4a9h ;     -     ;     0e49fh
; select to execute from page 1 disk area
link1:      ld          a,81h
            jr          linkx
```

```
; select to execute from page 2 disk area
link2:      ld          a,82h
            jr          linkx
; select to execute from page 3 disk area
link3:      ld          a,83h
linkx:      ld          hl,801h
            ld          (bspage),a
            ld          (pageflag),a
            ld          (bofa),hl
            call        clpage
            jp          zrun+3
            end
```

Demonstration program PROG1.

```
10 REM Demonstration program 1 (Must be in Page 1)
20 REM Set up the machine code link program at F605h (on 9G08 machine)
30 POKE 62981,62,129,24,6,62,130,24,2,62,131,33,1,8,50,241,252,50,146,249
40 POKE 63000,34,11,248,205,159,228,195,16,14
50 REM ***********************************************
60 REM CALL(62981) call to run program in page 1
70 REM CALL(62985) call to run program in page 2
80 REM CALL(62989) call to run program in page 3
90 REM ***********************************************

100 CLS:REM clear the screen
105 ON POWER RESUME:REM place machine in cont mode
110 PRINT"*****  PROGRAM 1 RUNNING  *****"
120 IF a(1)=55 THEN GOTO 1000 ELSE a(1)=55

130 REM initialise the variables for page 1
140 DIMa$(10,10)
150 FOR x = 1 TO 10
160 a$(x)="1"+JSR$(x,9)
170 NEXT
180 b$="This is program 1"

1000 REM demonstrate the variables for program 1 exist
1010 FOR x=1 TO 10:PRINT a$(x):NEXT
1020 PRINT b$
1030 PRINT"use looking keys to inspect data"
1040 PRINT"Press enter to continue to next program"
1050 INCHARa:IFa<>13 THEN 1050

1060 REM lets goto program 2
1070 z=CALL(62985):REM execute call to goto program 2
1080 END
```

Explanation:

Lines 30-40 load machine code into common ram
Line 100 clears the screen
Line 105 sets program into cont mode

Lines 120-180 initialise program 1 variables for inspection
Line 1000 displays variables for inspection
Line 1070 goes to program 2

Demonstration program PROG2:

```
10 REM Demonstration program 2 (Must be in Page 2)
20 REM Set up the machine code link program at F605h (on 9G08 machine)
30 POKE 62981,62,129,24,6,62,130,24,2,62,131,33,1,8,50,241,252,50,146,249
40 POKE 63000,34,11,248,205,159,228,195,16,14
50 REM ***********************************************
60 REM CALL(62981) call to run program in page 1
70 REM CALL(62985) call to run program in page 2
80 REM CALL(62989) call to run program in page 3
90 REM ***********************************************

100 CLS:REM clear the screen
105 ON POWER RESUME:REM place machine in cont mode
110 PRINT"*****  PROGRAM 2 RUNNING  *****"
120 IF a(2)=55 THEN GOTO 1000 else a(2)=55

130 REM initialise the variables for page 2
140 DIMc$(10,10)
150 FOR x = 1 TO 10
160 c$(x)="2"+JSR$(x,9)
170 NEXT
180 d$="This is program 2"

1000 REM demonstrate the variables for program 2 exist
1010 FOR x=1 TO 10:PRINT c$(x):NEXT
1020 PRINT d$
1030 PRINT"use looking keys to inspect data"
1040 PRINT"Press enter to continue to next program"
1050 INCHARa:IFa<>13 THEN 1050

1060 REM lets GOTO program 3
1070 z=CALL(62989):REM execute call to goto program 3
1080 END
```

Explanation:

Lines 30-40 load machine code into common ram
Line 100 clears the screen
Line 105 sets program into cont mode
Lines 120-180 initialise program 2 variables for inspection
Line 1000 displays variables for inspection
Line 1070 goes to program 3

Demonstration program PROG3:

```
10 REM Demonstration program 3 (Must be in Page 3)
20 REM Set up the machine code link program at F605h (on 9G08 machine)
30 POKE 62981,62,129,24,6,62,130,24,2,62,131,33,1,8,50,241,252,50,146,249
40 POKE 63000,34,11,248,205,159,228,195,10,14

50 REM ******************************************
60 REM CALL(62981) call to run program in page 1
70 REM CALL(62985) call to run program in page 2
80 REM CALL(62989) call to run program in page 3
90 REM ******************************************

100 CLS:REM clear the screen
105 ON POWER RESUME:REM place machine in cont mode
110 PRINT"****  PROGRAM 3 RUNNING  ****"
120 IF a(3)=55 THEN GOTO 1000 else a(3)=55

130 REM initialise the variables for page 3
140 DIMe$(10,10)
150 FOR x = 1 TO 10
160 e$(x)="3"+JSR$(x,9)
170 NEXT
180 f$="This is program 3"

1000 REM demonstrate the variables for program 3 exist
1010 FOR x=1 TO 10:PRINT e$(x):NEXT
1020 PRINT f$
1030 PRINT"use looking keys to inspect data"
1040 PRINT"Press enter to continue to next program"
1050 INCHARa:IFa<>13 THEN 1050

1060 REM lets goto program 1
1070 z=CALL(62981):REM execute call to goto program 1
1080 END
```

Explanation:

Lines 30-40 load machine code into common ram
Line 100 clears the screen
Line 105 sets program into cont mode
Lines 120-180 initialise program 3 variables for inspection
Line 1000 displays variables for inspection
Line 1070 goes to program 1

Restrictions

Note that once the BASIC programs are loaded, it is possible to delete the DUMMY files if disk space is required. It is essential that the BASIC programs begin at the start of each disk space. On no account must a file be deleted or edited, as this may cause the file handler to re-locate the file in the disk space memory resulting in the processor not executing the expected code when the program is re-run.

Also, as the editor uses the TPA, any arrays and variables will be destroyed by editing a file. As the arrays and variables are stored in the TPA area, they are shared between all programs.

Take care to structure the BASIC programs so that no conflict occurs between variables in different programs, for example by two programs using the same variable for two different purposes.

It is recommended that the use of BASIC statements which are latent in the Interpreter's execution and only become activated when a particular event occurs be avoided. These include:

```
ON COMMS GOTO
ON COM GOSUB
ON ERROR GOTO
ON POWER GOTO
ON BREAK GOTO
```

as the Interpreter does not act on these statements immediately, but sets up vectors within itself to cause execution to occur at a given line number when the particular event occurs. As paging techniques are used in this application, it is possible that when the event occurs, the program which set the event trap may not itself be under execution.

Consequently, when the event occurs, execution may go to a line number which may not exist, or may contain statements which are totally different to those expected for that function. It is advisable that if such statements are necessary, program switching is not employed until the programmer is absolutely sure that such events will no longer occur.

10.18.4 Chaining Programs

Chaining allows BASIC and machine code programs to call each other, but, unlike the program switching described earlier, calling a program loads it into the TPA, variables are not shared and the files can be anywhere on the RAM disk.

This technique requires the following simple machine code routine:

```
RUNCOM  EQU     0ADD3H  ;9G06
RUNCOM  EQU     0AE2BH  ;9G08
STACK   EQU     0F77FH

LINK:   LD      SP,STACK
        RST     20H
        DC      B(RUNCOM)
```

This routine must be loaded into the common RAM, for example into the user RAM area starting at location USER (F605H, 62981 decimal). This can be achieved by the following command:

```
POKE 62981,49,127,247,231,211,173 (ENTER)
```

To chain another program, you must store the complete filename followed by a CR in the operating system's DMA buffer (FCDMA+2: E082H or 57474 decimal) and then call the above routine. Note that the filename must include the file extension.

For example, to chain to a program called T1.HBA, you would use:

```
40 REM set up routine at USER
50 POKE 62981,49,127,247,231,211,173
60 REM set up file name
70 POKE 57474,84,49,46,72,66,65,13
80 REM chain file
90 A=CALL(62981)
```

Note that line 70 pokes the filename T1.HBA, including the " " and the CR.

You may use a similar technique to execute a .COM file from BASIC. For example, to execute X.COM, modify line 70 above to read:

```
70 POKE 57474,88,46,67,79,77,13
```

For further details, see Husky Technical Note TN200.

10.19 'Alarm Clock' Feature

In applications such as data logging, it is useful for the **Hunter 2** to be able to wake up at a specific time of day and perform an action such as take a reading.

The **Hunter 2** real-time clock can be programmed to generate an interrupt at intervals of 0.5, 5 or 60 seconds. If the **Hunter 2** is powered off, the interrupt will turn it on.

Using the BASIC keywords POWER, TIME$ and DATE$, you can write a simple program to check the date or time on power up, and decide whether to turn off again, or perform some other action (such as taking a reading or sending data to head office via a modem).

Consider the program:

```
10 OUT 79,2' SET CLOCK TO WAKE UP IN 5 SEC TIME
20 IF MID$ (TIME$,1,5) = "23:02" THEN 100
30 POWER OFF
100 REM TAKE READING
110 INPUT A
etc...
```

Line 10 programs the real time clock to generate an interrupt after a 5 second interval, which turns **Hunter 2** on if it is turned off. (OUT 79,1 programs a period of 0.5 seconds. OUT 79,2 programs a period of 5 seconds. OUT 79,4 programs a period of 60 seconds.)

Line 20 checks to see if the right time has been reached.

Line 30 turns off the **Hunter 2** and sets a flag to cause the program to execute from the start on power up. The **Hunter 2**, therefore, will power up and re-program the clock interrupt at the period determined by the OUT statement, on each clock interrupt.

When the preset time is reached, program execution jumps to line 100, which represents the logging phase of the program.

10.20 Errors and Warnings

Hunter 2's BASIC Interpreter will display an error message if an error occurs while a program is running and error trapping is not in force (see ON ERROR GOTO in Chapter 11).

The messages have three characters, generally with a line number. The meaning may be found with reference to the following table. They are aimed at experienced programmers.

Each error has an error number. This is the value returned by ERR and is useful during error trapping.

Message	No.	Meaning
ARG	0	ARGUMENT ERROR: A function has been called with an out of range argument.
STX	1	SYNTAX ERROR. Incorrect syntax in a BASIC statement.
CSK	2	CONTROL STACK ERROR. Incorrect FOR..NEXT; WHILE..WEND or GOSUB..RETURN construction.
SSK	3	SYSTEM STACK ERROR: Attempt to POP a non-existent variable.
DIN	4	DIRECT INPUT ERROR: Use of a BASIC statement which has no meaning as a direct command, eg NEXT
DIM	5	DIMENSION ERROR. Attempt to redefine an existing variable.
FP	6	FLOATING POINT ERROR: Illegal maths operation, eg B=A/C where C=0.
LNo	7	LINE NUMBER ERROR: Reference to a non-existent line number
SQR	8	NEGATIVE SQUARE ROOT: The argument to the SQR function is negative (the square root of a negative number is undefined).
MAG	9	MAGNITUDE ERROR: Reference to an array element larger than the array dimension.
RD	10	READ ERROR. The data statement is either invalid or non-existent.
MEM	11	STORAGE OVERFLOW· Variable storage or array overflowed available memory

Message	No.	Meaning
ARY	12	ARRAY ERROR: Use of an element in an undefined array.
BAS	13	BASIC SOURCE ERROR: Attempt to run a non-BASIC program.
HLP	14	HELP SOURCE ERROR: The HELP statement has an illegal line number reference.
TYP	15	TYPE MISMATCH ERROR. Received string data type instead of numeric.
STR	16	STRING COMPLEXITY ERROR: Insufficient space to manipulate a complex string expression.
RES	17	RESUME WITHOUT ERROR. RESUME executed when not in an error-handling routine.
FNF	18	FILE NOT FOUND: Reference to a non-existent file.
IFN	19	ILLEGAL FILE NAME: Filename does not conform to standards of file names.
FOP	20	FILE ALREADY OPEN: Attempt to open a file which is already open.
FNO	21	FILE NUMBER ERROR. File number greater than MAXFILES.
FCL	22	FILE CLOSED ERROR. Attempt to access a file which is not open.
FOO	23	FILE OPEN FOR OUTPUT ERROR. Attempt to read from a file which is open for output.
FOI	24	FILE OPEN FOR INPUT ERROR. Attempt to write to a file which is open for input.
WWD	25	WHILE/WEND ERROR. WHILE without WEND or WEND without WHILE.
DSK	26	DISK ACCESS ERROR. Disk access failed, for example, a write attempt might fail due to lack of disk space.
SFE	27	SYSTEM FILE ERROR: An attempt has been made to use the wand system files, and the Interpreter has either found the file is not loaded, or the file has been incorrectly loaded.
IPE	28	INPUT PAST END· An attempt has been made to input data beyond the end of a file which has been opened for INPUT
---	29-255	USER DEFINED ERRORS

Chapter 11 Reference to Hunter BASIC

This chapter lists the main **Hunter** BASIC keywords, in alphabetical order.

For details of the numeric, string, logical and relational operators (including NOT, AND, OR, XOR, EQV and IMP), see section 10.2.2.

11.1 ABS

11.1.1 Return absolute value of the argument

Syntax

ABS(N)

Where N can be a variable, number or result of a numeric expression.

Examples

Y = ABS(-12.345) returns Y = 12.345
Y = ABS(-0.5) returns Y = 0.5
Y = ABS(0.5) returns Y = 0.5

or if V = -27·
Y = ABS(V) returns Y = 27
Y = ABS(V-50) returns Y = 77

Remarks

The absolute value returned is always positive.

11.2 ARG

11.2.1 Pass parameters to machine code subroutine

Syntax

D = ARG(N)

Where D is a Dummy Variable and N is the integer value to pass

Examples

P = ARG(10) Passes 10
P = ARG(V) Passes the current value of V

Remarks

ARG loads the microprocessor's E and C registers.

They are loaded with the higher and lower byte portions of a 16 bit representation of the ARG argument.

For example if V in the example above was 4100 then:

E = 16 C = 4

(because 16x256 + 4 = 4100)

This function is useful when making CP/M system calls to set the C register before a call to location 5. See section 10.11.

11.3 ASC

11.3.1 Return decimal ASCII code of string

Syntax

ASC("STRING")

Examples

```
V=ASC(X$(3))
PRINT ASC("A")
```

will print 65

```
10 T$="AB"
20 PRINT ASC(T$)
```

will also print 65, since "A" is the first character.

Remarks

The string argument must be enclosed in parentheses.

The value returned is for the first character of the string. For a list of ASCII codes used with **Hunter** BASIC, see Appendix III.

ASC is particularly useful for detecting special characters like control codes, and distinguishing them from digits 0-9 or characters A-Z.

11.4 ATN

11.4.1 Return Arc-tangent

Syntax

ATN(N)

Examples

A = ATN(0.6009)

Sets A equal to 0.541071 Radians, ie 31 degrees

or

B = ATN(V)

sets B equal to ARCTAN of variable V

ATN is accurate to at least 10 decimal places in version 9G08, and less accurate in previous versions.

Remarks

The resultant angle is specified in radians.

To convert to degrees:

A = ATN(V)*180/PI

In the example above, PRINT (ATN(0.6009)*360/(2*PI)) returns 31.0011.

11.5 BEEP

11.5.1 Generate a beep

Syntax

BEEP

Example

```
10 FOR I = 1 TO 100
20 BEEP:NEXT I
```

Remarks

This function is equivalent to OPCHR 7 and produces a short tone from the sound generator For a more varied sound use SOUND

11.6 CALL

11.6.1 Call a machine code subroutine

Syntax

D = CALL(N)

where D is a variable and N is the address to call

Examples

10 A = CALL(0)
20 B = CALL(V)
Where V is the address to be called

Remarks

CALL switches execution to the program located at the location specified. It can be used to execute a user-supplied machine code subroutine or a routine in **Hunter 2**'s operating system. Operating system routines are described in Part 4.

The call in line 10 causes a restart of the **Hunter 2**'s software.

WARNING: Use CALL only as indicated in section 10.11. Use of unspecified calls may crash **Hunter 2**, resulting in loss of data, etc.

When control has passed back to BASIC from a machine code program, then D = the final number stored in the Z80 accumulator when the program returns.

11.7 CHAR

11.7.1 Select character set

Syntax

CHAR n[,I]

Where $0 < n < 4$

Examples

CHAR 2,I

This will output double width text in inverse video if $I > 0$.

Remarks

In graphics mode only there are five different character sets, with CHAR 0 being the default set.

I is the inverse parameter If it is zero then the characters are normal. If it is non zero then inverse video is displayed.

CHAR 0 produces a small 7 x 5 pixel character set defaulted to after a SCREEN 1 statement.

CHAR 1 is the standard 9 x 7 character set.

CHAR 2 are double width characters.

CHAR 3 are double height characters.

CHAR 4 are double height and double width characters.

NOTE: Each time the character size is changed, the cursor is moved to the upper left-hand corner

CHAR has no effect in text mode.

11.8 CHR$

11.8.1 Return character corresponding to ASCII code

Syntax

CHR$(N)

Where N is between 0 and 255.

Examples

```
10 DIM A$(10,10)
20 A$ = CHR$(34):REM see note below
30 PRINT "HE SAID,";A$;"HELLO";A$
40 B=7
50 PRINT CHR$(B):REM BEEP
60 PRINT CHR$(1):REM THIS CLEARS THE SCREEN
```

The above will print:

HE SAID, "HELLO"

then beep, and then clear the screen.

Remarks

CHR$ performs the inverse of ASC. It returns the single character string specified by the value of N, which may be an expression.

NOTE: The character " cannot be embedded in strings directly since it acts as a delimiter, but can be specified by using CHR$(34) as shown in line 20 of the example. (The ASCII code for " is 34.)

11.9 CIRCLE

11.9.1 Draw a circle

Syntax

CIRCLE [(X,Y)] ,r[,n]

Where $0<X<239$ and $0<Y<63$

Example

```
10 FOR I = 20 TO 220 STEP 40
20 CIRCLE (I,32),20
30 NEXT
```

This will plot six circles across the display.

Remarks

The optional parameter n will plot the circle in black if it is odd, or erase the points if it is even. When omitted, it defaults to odd. The co-ordinates may also be omitted, and the circle is then centred on the current graphic cursor position. Circles plotted off the edge of the screen will produce wrap around, Modulo 256.

The CIRCLE command range checks the x-coordinate, y-coordinate and radius to be less than 256. Since the **Hunter 2** screen in graphics mode can only display 0-239 and 0-63 in the x and y directions respectively, there are occasions when the expected circle does not appear on the screen.

11.10 CLEAR

11.10.1 Set all variables to zero.

Syntax

CLEAR

Examples

CLEAR

resets all variables.

Remarks

Variables are also set to zero if the program is altered or added to. Variable contents are otherwise maintained indefinitely by **Hunter 2**, as described earlier. Clear also resets the FOR..NEXT loop stack and arithmetic stack.

11.11 CLOSE

11.11.1 Close files

Syntax

CLOSE [#file 1,#file 2,.....#file X]

Examples

CLOSE #4,#5

Will close file numbers 4 and 5.

Remarks

If CLOSE is specified with no parameters all files that have been OPENed are CLOSEd.

When the CLOSE command is executed, **Hunter 2**'s operating system automatically performs a checksum generation on the file and stores the value in the file directory for checking the next time the file is opened.

Failure to CLOSE the file in the correct manner will result in a 'No file' error message when an attempt is made to OPEN the file at a later date.

When a file is closed, any data in the buffer not already written to the file will be saved. This means that there must be space available in the **Hunter 2** for a file to be closed correctly. It is therefore recommended that the following method be used to ensure data integrity in all application programs where the above problem may arise.

```
10 OPEN "EXPAN.DTA"FOR OUTPUT AS 1
20 WRITE #1,A:CLOSE
```

This reserves an area of memory in the disk space. Using ON ERROR to trap a "disk full" error, the handling routine could simply be:

```
1000 KILL"EXPAN.DTA"
1010 CLOSE
```

This ensures that there will always be enough room to close all files.

11.12 CLS

11.12.1 Clear display screen

Syntax

CLS

Example

100 CLS

This clears the screen ready for text or graphics output.

Remarks

In either graphics or text mode, CLS will clear the LCD display. It has a similar effect to OPCHR 1. In graphics mode, the screen may also be cleared (and the character set initialised to CHAR 0) with SCREEN 1.

NOTE: Only the screen currently in use (text or graphics) is cleared by CLS.

11.13 COM

11.13.1 Control communications interrupt handling

Syntax

COM ON
COM OFF
COM STOP

These statements must be initialised with an ON COM statement.

Example

```
10 ON COM GOSUB 1000
20 COM ON
1000 REM TRAP ROUTINE
1010 LINPUT A$ ...
1990 RETURN
```

Remarks

These statements decide what action is taken on receiving communications. The interrupt occurs on the first character

COM ON	When executed, allows trapping of communications by the ON COM statement. If the previous state was COM OFF then the communications receive buffer is cleared by this statement.
COM OFF	The trapping is deactivated and all communications interrupts are ignored.
COM STOP	Communications interrupts are noted but not acted upon by the ON COM statement until a COM ON is executed.

11.14 CONT

11.14.1 Continue execution after STOP

Syntax

CONT [Line number]

Line number is optional and allows the program to continue from a different line number

Examples

CONT continue from next line

or

CONT 1000 continue from line 1000

Remarks

Allows a program to continue execution from the line following the occurrence of:

a) detection of the break key (ESC or CTL/FN C)

b) execution of a STOP statement

c) execution of an END statement

d) detection of a syntax error

By specifying a line number argument, you can make execution continue from that line.

If the program stopped part way through a multi-statement line, execution will begin from the start of the next line.

11.15 COS

11.15.1 Return the Cosine of the argument

Syntax

COS(angle)

Where angle is in radians.

Examples

X = COS(Y) sets X equal to the cosine of Y

R = COS(5) sets R equal to the cosine of 5 radians.

Remarks

For angles in degrees use:

X=COS(A/180*PI)

COS is accurate to 12 decimal places.

11.16 CRT

11.16.1 Use serial port as keyboard and screen

Syntax

CRT

Examples

CRT

Remarks

CRT redirects keyboard and display functions via the RS232 port.

The command allows you to use any standard VDU with the **Hunter 2** instead of its own keyboard and screen. This may be useful for entering programs quickly or typing characters that are not on **Hunter 2**'s keyboard.

Any standard VDU may be used. Baud rates and other parameters are set on **Hunter 2** using 'Initialise Communications'.

Remember that a 'crossed' RS232 lead is needed with GND, RXD and TXD circuits installed.

11.17 CUROFF/CURON

11.17.1 Turn cursor off and on

Syntax

CUROFF
CURON

Example

CUROFF

To remove the cursor when display in use.

CURON

This will turn the cursor on.

Remarks

Hunter 2 defaults to CURON following power up.

11.18 DATA

11.18.1 Store data for use by READ

Syntax

DATA *item-1, item-2....., item-n*

Examples

```
100 DATA 12,A1,25,SIN(0.5),MID$(A$,3,4),99
110 DATA 5,6,7,8,6*5,"HELLO"
```

After reading the value 99 the READ will take 5, and so on, until every data value is used. This form of data storage is not affected by CLEAR or program line changes which will erase variables. To return to the first value in a DATA list use the RESTORE statement.

A data statement must appear on a program line by itself. It cannot form part of a multi-statement line.

Remark

The arguments for DATA statements can be either numerical values, variables or expressions. Multiple arguments are separated by commas.

String variables may be used in DATA statements. String constants must be enclosed in quotes.

Care must be taken to ensure that variables in a READ statement are of the same type as the data being read into them.

11.19 DATE$

11.19.1 Return or set the date

Syntax

A$ = DATE$
or
DATE$ = B$

Example

```
10 DATE$ = "5/12/86"
20 PRINT DATE$
```

Remarks

DATE$ returns a ten character string of the format MM-DD-YYYY. As in the example, when DATE$ is used as a variable leading zeros may be omitted and the century defaults to 19yy

DATE$ will accept / or or - delimiters to set the date, but will always print the date using - as a delimiter

11.20 DAY$

11.20.1 Return or set the day

Syntax

A$ = DAY$
or
DAY$ = A$

Examples

DAY$ = "MON"

will set DAY$ to MON

100 PRINT DAY$

will output:

Mon

Remarks

The day string may be input in upper or lower case.

Acceptable days are:

MON
TUE
WED
THU
FRI
SAT
SUN

DAY$ always outputs the day with an initial capital letter (Mon, Tue, etc).

11.21 DEFSEG

11.21.1 Define RAM page for machine code access

Syntax

DEFSEG = *page-no*

Examples

DEFSEG = 1

sets RAM page to 1

DEFSEG may also be used as part of an expression:

PRINT DEFSEG*3

Remarks

DEFSEG selects which page of RAM will be addressed by subsequent CALL, PEEK or POKE commands.

The *page-no* is an expression from 0 to the last page in the system. When **Hunter 2** is powered up, DEFSEG is set to 0. This is the TPA or workspace, most often used for machine code programs called from BASIC.

After a program error, DEFSEG is set back to 0.

If you use DEFSEG to change the RAM page as a direct statement, take care in subsequent direct statements to avoid corrupting the memory or disk space.

Note: If DEFSEG is executed as a direct statement, you should either turn off and on again or set DEFSEG back to 0 before attempting to use other functions or facilities.

11.22 DIM

11.22.1 Set up arrays

Syntax

DIM *variable-name*(*array-size*)

DIM *double-precision-variable-name*(*array-size*)

DIM *string-variable-name*(*array-size*, *element-size*)

where *array-size* and *element-size* are positive numeric expressions.

Examples

DIM A(25)

defines an array variable A of 26 elements, including the element A(0).

DIM J4(100),J5(100),Q(1000),J(N)

defines multiple arrays. N is a previously defined variable.

Remarks

Three kinds of array can be set up by a DIM statement in **Hunter** BASIC:

* Simple variables
* Double precision variables
* Strings

NOTE: The maximum number of elements in a single array is 16,384.

All array types have a range from 0 (zero) to the limit defined in the DIM statement.

Default values for numeric arrays are 10 elements. Strings default to one element of 20 characters, larger strings have to be dimensioned.

Arrays can be allocated to any combination of variable names.

Double precision arrays are defined in the same way except that the array name ends with a "!" Each element has a 14-digit size.

For example: DIM A!(25) and DIM AZ!(32) define double precision arrays.

Space for string arrays is reserved in **Hunter 2** memory using a DIM statement followed by string name, the number of elements in the array, and the maximum string length required. For example:

DIM A$(25,10)

defines a string array with 26 elements each of maximum length of 10 characters.

Each character in a string requires 1 byte of storage.

You are not allowed to redefine an array

Hunter 2 retains variable (including array) definitions until another program is loaded, or CLEAR or NEW is used. If a program will be used by running it without reloading it each time, you must make sure that the DIM statements are not executed each time.

One solution is to initialise arrays once, by using direct DIM commands.

A more general technique is to set a switch in the application software to indicate whether the program has been run before. If the switch is a variable which is made non zero when the program is first run, the fact that any modifications or use of the CLEAR statement will clear the value to zero can be used to determine whether the arrays should be redefined or not.

```
10 If A = 1 THEN GO TO 100
20 DIM J(50)...
90 A = 1
100 REM user program starts here...
```

The variable A could instead be used as a counter to indicate the number of times that the program has been run.

11.23 EDIT

11.23.1 Invoke the Hunter 2 text editor

Syntax

EDIT [*filename*]

Examples

EDIT

The editor is invoked. If a BASIC program is currently loaded in the workspace, the editor will operate on it. If no program is loaded, entry of new text or program may proceed.

EDIT "PROG1"

Program PROG1.HBA is first loaded into the workspace. EDIT is then invoked.

Remarks

For a full description of the editor see Part 2.

To exit EDIT, either save the file (CTL/FN 6, followed by filename) or Exit (CTL/FN 8) The edited version of the program will remain in the workspace.

NOTE: Following 'Save' or 'Exit' commands, the editor will display a message:

** Wait

A few seconds may elapse while the editor converts the BASIC source text back into the more compact (tokenised) format of the .HBA file.

Only files with the .HBA suffix are treated in this way. When invoked from BASIC, the editor assumes .HBA as a default suffix.

11.24 END

11.24.1 Terminate user program execution

Syntax

END

Examples

1000 END

when executed, displays:

READY

and returns control to the BASIC command level.

Remarks

END does not require an argument.

This is normally the last line of a program although END may also be used in the middle of a program to terminate it. If there is no END at the end of the program, BASIC will terminate as if there were one.

User programs will in general not END but retain control over **Hunter 2** operation.

11.25 EOF

11.25.1 Check for end of file

Syntax

EOF(*file-no*)

Examples

```
10 OPEN "DATA.TXT" FOR INPUT AS #1
20 INPUT#1,A,B,C
30 PRINT A,B,C
40 IF EOF(1)THEN CLOSE #1 ELSE GOTO 20
50 PRINT"END OF FILE":END
```

Remarks

EOF returns 1 if end of file has been reached, or 0 if more data is in the file.

Programs should not try to read beyond the end of a file since an error will occur.

The file DATA.TXT used in this example, could be created by a BASIC program of the form:

```
100 OPEN "DATA.TXT"FOR OUTPUT AS #1
110 WRITE#1,123,235,345
120 CLOSE#1
130 END
```

This will ensure that the correct end of file marker is written to the file.

11.26 ERR and ERL

11.26.1 Return error code and line number

Syntax

```
X = ERR
Y = ERL
```

Example

```
10 GOTO 50
20 A = ERR
30 PRINT "Error Number";A;"at line";ERL
50 ON ERROR GOTO 20
60 B = A/0
70 STOP
```

This example prints:

Error Number 6 at line 60

due to the floating point error caused by an attempt to divide by zero.

Remarks

ERR and ERL will always contain the codes of the last error, and will often be used in conjunction with ERROR.

These functions are often used to handle errors in communications which have been trapped by using ON ERROR GOTO.

See section 10.20 for the error numbers.

ERR defaults to zero until an error occurs.

11.27 ERROR

11.27.1 Simulate a BASIC error

Syntax

ERROR *n*

where $0 < n < 255$

Example

```
10 INPUTa,b
20 ONERRORGOTO100
30 IFa>bTHENERROR156
40 GOTO10
100 IFERR = 156THENPRINT"a>b error"
110 RESUME 10
```

Remarks

ERROR allows you to generate BASIC errors, including user defined codes. For a list of BASIC error messages, see section 10.20.

The ERROR statement neatly traps errors and forces the printing of a BASIC error message. If the error number is undefined in BASIC, then processing can be forced to restart at another line to print a user defined message. When ERROR is set to n, this number may be recovered with ERR, and the line number with ERL.

After a forced user error trap, a suitable RESUME command must be executed to ensure correct program operation.

11.28 EXP

11.28.1 Return the value of e raised to the power of X

Syntax

EXP(*n*)
Where *n* is a numeric expression in the range -290 to +290

Examples

A!(0) = EXP(3)
sets A!(0) to e to the third power

B=EXP(N)
sets B to e to the power of N

Remarks

EXP performs the reverse function to the LN function (ie it performs as an 'anti-logarithm').

If X is outside the allowed range a 'MAG Error' will occur

e is defined as 2.7182818284590

11.29 FILES

11.29.1 List user files

Syntax

FILES

Example

This program shows how to use FILES to help a user select the right file:

```
6030 CLS:PRINT"Please enter the file you wish to select:"
6040 LOCATE0,1:PRINT"(Use shift arrow keys to scroll)"
6050 LOCATE0,3:FILES
6060 LOCATE0,2:INPUT"enter filename ";a$
```


11.30 FIX

11.30.1 Return integer version of argument

Syntax

A=FIX(*n*)

Examples

A=FIX(12.37) sets A = 12

A=FIX(-5.95) sets A=-5

Remarks

FIX converts its argument to an integer by removing any fractional part. It should not be confused with INT which converts its argument to the next lower integer

11.31 FOR

11.31.1 Repeat a sequence of instructions a number of times

Syntax

FOR N = *start* TO *end* STEP *step*

NEXT N

Where N is a numeric variable, *start* is a numeric start value, *end* is a numeric end value and *step* is an increment/decrement value.

start, *end* and *step* can be numbers, variables or expressions.

Examples

```
10 FOR A = 1 TO 100 STEP 10

100 NEXT A
```

causes the value of A to equal 1,11,21 etc., for each execution of the loop.

To obtain a decrementing count, use a negative value with STEP·

```
1000 FOR Z1=91 TO 10 STEP -.1
.
1050 NEXT Z1
```

Remarks

FOR will execute the statements between it and the NEXT statement a number of times calculated as follows: (*end-start* + 2)/*step*

You should exit a FOR..NEXT loop via the NEXT statement, not by jumping out of the loop (using GOTO). Alternatively, if the loop is within a subroutine,

you may leave the loop and the subroutine at the same time, by using RETURN. (If you do jump out of a loop, information is left on BASIC's 'control stack', that will probably cause a 'CSK' error later)

11.32 FRE

11.32.1 Return the number of bytes free for BASIC

Syntax

FRE(0)

0 is a dummy argument (any numeric will do)

Examples

PRINT FRE(0)

displays amount of free space in execution memory

or

A = FRE(0)

Sets A = amount of free space.

Remarks

If a program has not been RUN then the variables will not have been assigned space, so that FRE(0) allows you to calculate the program size. Once the program has run, FRE(0) will allow you to calculate the amount of space used by variables.

FRE(0) can be used to size an array to the maximum allowed by the memory ('auto-sizing'), like this:

DIM A((FRE(0)-10240)/5)

This autosizes A leaving 10240 bytes (10K) free for other variables which have not yet been defined. The number "5" is the space that each element of the A array takes up. Remember to leave enough space for all simple variables.

11.33 GOSUB

11.33.1 Jump to a BASIC subroutine

Syntax

GOSUB *line-number*
where *line-number* may be a number or numeric expression

Examples

GOSUB 150

causes program execution to jump to a subroutine located at 150.

The argument can also be an argument or expression, although this is not generally regarded as good programming practice. For example:

GOSUB SI

or

GOSUB SI + 100*A

This is illustrated by the program sequence:

```
10 INPUT A
20 SI = 1000
30 GOSUB SI + 100*A
40 STOP
1000 PRINT "THIS IS SUBROUTINE 1"
1010 RETURN
1100 PRINT "THIS IS SUBROUTINE 2"
1110 RETURN
etc.
```

If 0 is entered in line 10, the program will execute subroutine 1. If 1 is entered, subroutine 2 will be executed.

Remarks

A subroutine must always be terminated by a RETURN statement. Program execution then continues from the statement immediately after the GOSUB. GOSUB always requires a line number as an argument.

GOSUB may not be used as a direct statement.

11.34 GOTO

11.34.1 Branch unconditionally to a specified line

Syntax

GOTO *line-number*
where *line-number* is a number or numeric expression

Examples

GOTO 50

causes program execution to commence immediately at line 50.

10 GOTO 100

causes program execution to skip from line 10 to line 100.

The line number may also be a variable or an expression, although this is not generally regarded as good programming practice. For example:

GOTO A or GOTO A*100

are equally valid.

The program sequence:

```
20 ZB = 150
30 GOTO ZB
150 PRINT "THIS IS LINE";ZB
```

would display: THIS IS LINE 150

Remarks

GOTO is always used with an argument (line number). If used as an immediate command, GOTO will cause program execution to commence from the line number specified. If used in a program, GOTO will cause program execution to skip to the line number specified.

If the line specified is not present in memory **Hunter 2** will display:

LNo Error in line 150

(If you use GOTO as a direct command, the line number specified in the error message will be meaningless.)

11.35 HELP

11.35.1 Define location of help text

Syntax

HELP *line-number*
where *line-number* is a number or numeric expression

Examples

```
1010 REM THIS IS A PIECE OF HELP TEXT
1020 REM INTENDED TO ASSIST THE OPERATOR
1030 REM IN THE USE OF HUNTER 2
1040 REM PLEASE PRESS HELP TO RETURN
1050 REM TO PROGRAM.
10000 HELP 1010
```

Execution of line 10000 will cause the block of remarks to be used as the HELP text when [HELP] is pressed.

Remarks

HELP requires an argument which must be a valid line number. The referenced line must be a REM statement.

When the operator presses HELP, text from the referenced REM is placed on the screen. If there is more HELP text than will fit onto the screen, the operator can scroll through it by using the ↑ and ↓ keys.

The HELP text start may be changed by executing another HELP statement.

[HELP] is normally disabled. Execution of HELP causes the HELP code to be loaded into the keyboard table, therefore enabling the use of [HELP]. Once the HELP statement has been executed, it is not possible to disable [HELP] except by deleting the HELP code from the keyboard map.

On power up, [HELP] will be disabled again, so the program should be written to execute the HELP statement again during program initialisation.

It is recommended that all the HELP text be written contiguously so that the operator may scroll through all of it. Scrolling stops when any non-REM line is encountered, so blocks of text can be easily partitioned if required.

NOTE: In graphics mode, [HELP] is ignored. Use [Ⓕ] [PWR] to turn off.

11.36 IF

11.36.1 Execute statements depending on an expression or comparison

Syntax

IF A [*operator* B] THEN *action1* [ELSE *action2*]

Where A and B are expressions, variables or constants of the same type (ie both numeric or both string) and *operator* is a logical operator: =, < , > etc. If only A is specified, it must be numeric.

action1 and *action2* can be one or more statements, or line numbers.

Examples

```
50 IF A = B THEN 100
60 IF A>B THEN PRINT "NO"
100 IF A=B THEN X=10 ELSE A=A+1:Y=1
2000 IF D AND NOT B THEN C=0:GOTO50 ELSE RETURN
```

action1 and *action2* may be a sequence of statements separated by colons. These will be executed in their entirety or not, depending on whether the expression evaluates to true.

```
IF A=6 THEN IF B>10 THEN A=0 ELSE ELSE A=A+1
```

Omitting the first ELSE will move processing straight on to the next line if the first condition (A=6) is false and cause A to be incremented if A=6 and B>10.

BASIC also permits nesting of IF statements. For example:

```
IF A= B THEN IF C=D THEN PRINT "FINISHED":GOTO 1000
```

String expressions may be tested for equality or inequality, for example:

```
IF A$ + B$ = "ABCDEAB" THEN 100
```

or

```
IF A=B AND B=C OR D=5 THEN PRINT "TRUE"
```

In this example "TRUE" will be printed if A=B and B=C, or if D=5.

However,

```
IF A=B AND B=C AND D=5 THEN PRINT "TRUE"
```

will only print "TRUE" if all three conditions are satisfied.

IF A THEN PRINT "TRUE"

will print "TRUE" if the variable A is non-zero. For example, if A=-19 then "TRUE" will be printed.

See section 10.2.2 for more details of logical and relational operators.

Remarks

The expression must evaluate to true (non-zero) or false (0). If the expression is true then *action1* is executed. If it is false then *action2* is executed.

If strings of unequal length are being compared, they will be considered to be equal if the string on the left is equal to or is the initial part of the string on the right. For example:

IF A$ = "ABC" and B$ = "ABCD"

then A$ = B$ is true, but B$ = A$ is not true

action1 and *action2* may include other conditional statements (nested), but care must be exercised in the positioning of ELSE to associate it with the correct IF Generally, each ELSE is associated with the most recent IF. You need not include extra ELSE statements at the end of the line if their dependent statements would be null ie have no action.

11.37 INCHR

11.37.1 Input a single character from the keyboard

Syntax

INCHR [@(*x*,*y*)] [*prompt-string*;] *numeric-variable*

where *prompt-string* is an optional string constant that will be displayed to prompt the operator and *x* and *y* are numeric expressions specifying the coordinates at which to print the prompt.

Examples

```
10 INCHR "DECIMAL VALUE";A
20 PRINT A
30 GOTO 10
```

displays:

```
DECIMAL VALUE ?A 65
DECIMAL VALUE ?B 66
etc.
```

Remarks

INCHR does not wait for [ENTER] to be pressed. A prompt message is displayed as with INPUT Only one variable may be entered. The variable is set to the decimal value of the character, as detailed in Appendix III.

A comma following the prompt string will suppress the prompt character '?'. This comma replaces the semi-colon in the syntax.

NOTE: If the operator presses [2ND], [HELP], [CTL/FN] or [PWR], they will have their normal effects. They will NOT be returned as characters.

[ESC] does not have its usual effect when typed in response to an INCHR statement. It returns a value of 27

11.38 INKEY

11.38.1 Scan the keyboard and return an ASCII code

Syntax

INKEY A

Examples

IF INKEY A = 65 THEN STOP

detects capital A and stops.

Remarks

The variable A is attributed the value of the key pressed. If no key is detected then the value returned is zero.

INKEY does not wait for a key to be pressed; execution continues immediately with the next statement.

INKEY provides a fast method for checking the keyboard status.

Repeated use of INKEY (eg in a loop) will decrease battery life. See section 10.14.

11.39 INKEY$

11.39.1 Scan the keyboard and return a character

Syntax

A$ = INKEY$

Example

```
10 A$ = INKEY$
20 IF A$ = "" THEN 40
30 PRINT A$
40 GOTO 10
```

Remarks

If a key has not been pressed then a null string is returned. If a normal key has been pressed, INKEY$ returns it as a single character string. If a function key has been pressed then a two character string is returned, consisting of the '^' character and the number of the key pressed. For example, pressing Function Key 1 will return the string "^1"

Any characters read via INKEY$ are not displayed. To show them you must print them explicitly, as in line 30 in the example.

INKEY$ may be used for example to wait until the user has read a screenful of information before displaying another, like this:

```
80 PRINT"last line of last screen"
90 A$=INKEY$: IF A$="" then 90
100 PRINT"First line of next screen"
```

Repeated use of INKEY$ (eg in a loop) will decrease battery life. See section 10.14.

11.40 INP

11.40.1 Input a byte from a port

Syntax

X = INP(*port*)

where *port* is a numeric constant variable or expression giving the port number.

Examples

A = INP(132)

Remarks

Refer to Part 4 for details of **Hunter 2**'s ports.

A 'port' is a term used at the hardware level of **Hunter 2.** It refers to the input/output addresses used by the NSC800.

11.41 INPUT

11.41.1 Input a line of data from the keyboard

Syntax

INPUT [@(*x*,*y*)] [,] [*prompt-string*;] *var-1*, *var-2*, ...*var-n*

where *x* and *y* are numeric expressions specifying the coordinates at which to print the prompt.

Examples

INPUT A

Displays the prompt "?" and inputs a numeric value from the keyboard and places it in the variable A.

INPUT"Parameters";A,B,C$

Displays the prompt "Parameters?" and then inputs values to A, B and C$.

A comma following the prompt string suppresses the question mark prompt following the prompt string. For example:

INPUT"Parameters",A,B,C$

Displays the prompt "Parameters"

To disable the prompt entirely, specify a null prompt string and use a comma:

INPUT"",A

A semi-colon prior to the prompt string will prevent the cursor moving to the next line when the user presses ENTER. This allows you to retain complete control of the cursor, which can be very useful if inputting from a menu or 'form' displayed on the screen (to prevent interference with the display), or inputting from the bottom line of the screen (to inhibit scrolling).

Remarks

The operator must terminate the input line by pressing ENTER.

INPUT cannot be used as a direct statement.

If INPUT's argument is a simple variable and an invalid entry is made, INPUT will prompt again with an error message. For example, if the INPUT statement were:

10 INPUT A

and the operator typed HELLO, BASIC would prompt with:

?Re-do from start
?

and wait for an acceptable entry

11.42 INPUT USING

11.42.1 Input and validate a line of data from the keyboard

Syntax

INPUT USING (*validation-string* [,*min*,*max* [,E]]) [*prompt-string*[, or ;]] *string-variable*

where *min* and *max* are numeric expressions or blank. They must lie in the range:
0 <= min < max
min < max < 96

NOTE: 'E' is only valid if *min* and *max* are present.

Examples

INPUT USING ("A999",2,4)S$

Displays a ? prompt and requires entry of a minimum of 2 characters and a maximum of 4. The first character must be a letter and the remaining characters must be decimal numbers (ie in the range 0-9). The input is stored in the string variable S$.

INPUT USING ("A999",,4)S$

The same as above, but the minimum defaults to 1.

INPUT USING ("A999",2,)S$

A minimum entry length of 2 and a default maximum of 95. Characters after the fourth are validated as wildcard characters.

INPUT USING ("A999")S$

Minimum and maximum default to 1 and 95 respectively.

INPUT USING (M$,3,6)S$

A minimum of 3 and a maximum of 6 characters are allowed. The entries are validated against the string expression M$. A '?' prompt will be displayed.

INPUT USING (M$,3,6)"ENTER",S$

As above, but with an 'ENTER' prompt.

INPUT USING (M$,3,6,E)"ENTER",S$

Same as above, but entry will terminate automatically after the sixth character has been entered.

Remarks

INPUT USING allows you to validate each character of the input as numeric, alphanumeric, etc., as it is entered. If a character fails validation, it is rejected with a beep.

You can also specify a required entry length, as a range. If the operator attempts to type too many characters, they will be rejected with a beep. If the operator presses [ENTER] when the entry is too short, this will be rejected with a beep.

The input validation string consists of defined validation characters either in the form of a string expression (A$, B4$ etc), or a string constant ("A99" etc) enclosed in quotations. The input validation string must be specified.

The optional *min* and *max* parameters define the minimum number and maximum number of characters required. If omitted, they default to a minimum of 1 and a maximum of 95. If the validation string is shorter than the maximum, the input characters for which there are no corresponding validation characters are validated against wildcard characters (see below). If min is 0 an input of 'CR' only is allowed.

[ENTER] terminates entry but will not be accepted if the number of characters is less than minimum.

When the number of characters exceeds maximum no more characters will be accepted and a warning tone will sound. The operator can use the → and ← keys to edit the entry before pressing [ENTER].

The E parameter causes the entry to terminate automatically once the maximum number of characters have been entered (the operator does not have to press [ENTER]).

CAUTION: If the input field extends over more than one **Hunter 2** screen line confusing results may occur in subsequent attempts to delete or edit characters.

To suppress the "?" prompt, use a comma after the prompt string. There must be a prompt string when using comma, therefore, to have no prompt at all it is necessary to use the null string "".

A string, enclosed by quotes, immediately following the validation expression will cause that string to act as the prompt.

→ and ← are enabled in order to allow the user to modify incorrect entries by simply overtyping them. [DEL] will delete the last character entered.

A-Z only
A-Z space
A-Z 0-9
A-Z 0-9 space
0-9 + - ,
0-9 only
Decimal point only
Wildcard

If you include other characters in the validation string, this will give a syntax error when the statement is executed.

Wildcard allows any character in the range space to DEL (ASCII 32 to 127), which includes all letters, numbers and punctuation symbols.

Control characters are not accepted.

11.43 INPUT#

11.43.1 Input data from a file

Syntax

INPUT#*file-no*, *var-1*, *var-2*...*var-n*

Examples

INPUT#1,A,B$,C$

This will input data to the variables A, B$ and C$ from file number 1.

Remarks

The variables specified in the INPUT statement should match the type of data stored in the file.

The End of File function EOF should be used to check if the end of the file has been reached. See section 11.25 for examples.

11.44 INSTR

11.44.1 Search for substring within a string

Syntax

Y = INSTR ([*start*,]*string*,*substring*)
where 1 < *start* < 255

Example

```
10 A$="BASICS"
20 PRINT INSTR(A$,"S")
```

This program will print 3, since this is the position of the first occurrence of the letter S in "BASICS"

```
10 A$="BASICS"
20 PRINT INSTR(4,A$,"S")
```

This program will print 6, since this is the position of the first occurrence of the letter S when searching from the fourth character onwards in "BASICS"

```
10 A$="SUPER BASICS"
20 B$="SI"
30 PRINT INSTR(A$,B$)
```

This program will print 9, since this is the position of the first occurrence of "SI"

Remarks

If the substring does not occur in the string, then 0 is returned. The start is used to commence the search at a particular place within the string, it is optional, and if omitted defaults to 1.

Case is also important, ie "s" is not the same as "S"

11.45 INT

11.45.1 Round argument down to the next lower integer

Syntax

INT(N)
where N is a numeric expression

Examples

A = INT(123.456)

sets A to 123.

A = INT(-4.75)

sets A to -5.

Remarks

Negative numbers with a fractional part are rounded down ie in a negative direction.

FIX also converts its argument to integer, but by removing the decimal fraction part.

11.46 JSR$

11.46.1 Create right justified string from numeric

Syntax

JSR$(A,N)
where A is the numeric argument to convert and N is the length of string to create

Examples

If A = 12.34 then:

JSR$(A,8) returns -0012.34
JSR$(A,5) returns -12.3

If A = 12.34 then:

JSR$(A,8) returns 00012.34
JSR$(A,5) returns 012.3

Remarks

If the number is negative, a leading - sign will be returned. If the number is positive, a leading 0 will always be returned, instead of a + sign.

The sign (0 or -) and decimal point each count as one character

If the length is insufficient to output the entire number it will be truncated without error For example, JSR$(12345,3) will return "012"

11.47 KEY

11.47.1 Control function keys

Syntax

KEY ON - enable function keys display

KEY OFF - disable function keys display

KEY LIST - list function keys

KEY *key,string* - redefine function key
where 1 < = *key* < = 8

KEY 0 - return function keys to default

Examples

KEY ON

Allows the function key display to be displayed on the bottom line of the display. This line is then not scrolled.

KEY OFF

Removes the function key display from the bottom line of the screen, allowing this line to be used like the other screen lines. It does not inhibit the use of function keys.

KEY LIST

This displays the entire string assignments of the eight function keys on the screen.

KEY *key,string*

Programs the specified function key with a string. The first four characters will be displayed on the bottom line of the screen if the function key display has been enabled. For example:

KEY1,"?X,Y"

will set function key 1 to print the values of the variables x and y when struck. Using the function keys is equivalent to using the keyboard - but faster

KEY 1, "?TIME$" + CHR$(13)

will display the time when you press function key 1.

Remarks

On entering BASIC, the function keys default to:

KEY			KEY		
	1.	File(s)		5:	Save(")
	2:	Run		6:	Load(")
	3:	Edit		7·	Kill(")
	4:	List		8:	Syst(em)

However, if auto-run on power up mode has been selected, the function keys are left as they were last programmed.

To return the function keys to these defaults, use the KEY 0 command.

To include a control code (such as CR, typed by ENTER) or a " in a function key string, use the CHR$ function. For example, CHR$(13) for CR.

11.48 KEY(n)

11.48.1 Control trapping of function keys

Syntax

KEY(*key*) ON - turn trapping on

KEY(*key*) OFF - turn trapping off

KEY(*key*) STOP - suspend trapping

where *key* is a numeric expression. If it is in the range 1 to 8 it refers to the corresponding function key. If it is in the range 11 to 14 it refers to a cursor key as follows:

key	Keystroke
11	CTL/FN +
12	CTL/FN +
13	CTL/FN +
14	CTL/FN +

Example

```
10 ON KEY (1) GOSUB 1000
20 KEY (1) ON

.
900 END 'FINISH OF MAIN ROUTINE
1000 REM FUNCTION KEY (1) TRAP ROUTINE

.
1990 RETURN
```

Remarks

The KEY (*key*) statement is used to trap function keys and cursor keys during program execution. It is initialised with the ON KEY (*key*) statement.

KEY (*key*) ON activates trapping of the specified key When that key is pressed, a GOSUB is automatically executed to the line specified in the corresponding ON KEY (*key*) statement. The normal action of the key is cancelled (for example, typing as function key's string).

The display of function key strings is not altered by these commands.

If the same key is pressed again while the subroutine is being processed, then the interrupt is stored until the program returns from the interrupt routine when the GOSUB will be executed again.

If a different key is pressed while one interrupt routine is being processed, then another GOSUB is immediately executed. Different key trapping routines may be nested, but each individual routine cannot be re-entered.

KEY (*key*) OFF deactivates the trapping mechanism so that the key has its usual effect. For example, a function key will type the string defined for that key (This will be ignored until the program fetches input from the keyboard, when it will be interpreted as normal text.)

KEY (*key*) STOP cancels the normal effect of the key and suspends handling of the key. The keystroke will however be remembered and handled if a KEY (*key*) ON statement is encountered for that key

11.49 KILL

11.49.1 Delete a file

Syntax

KILL *filename*

Examples

KILL "FIRST.HBA"
deletes FIRST.HBA

KILL A$
deletes the filename contained in A$

Remarks

To KILL a file, *filename* must be given either explicitly or may use the "?" wildcard. Variable names are accepted, but must contain actual filenames.

If the specified file does not exist, then no action is taken and no warning message issued.

If wildcards are used, then all files which conform to the format are erased. For example, if there are files called H.1 and H.2 then KILL "H.?" will delete both of them.

11.50 LEFT$

11.50.1 Return the first n characters of a string

Syntax

LEFT$(*string,n*)

where *string* is a string expression and *n* is the number of characters.

Examples

```
10 A$ ="ABCDEFG"
20 B$ = LEFT$(A$,3)
30 PRINT B$
```

prints ABC.

Remarks

LEFT$(A$,0) returns a null string ie ""

LEFT$(A$,50) returns ABCDEFG

11.51 LEN

11.51.1 Return the length of a string

Syntax

LEN (*string*)

where *string* is a string expression.

Examples

```
10 A$ = "ABCDEFG"
20 PRINT LEN (A$)
```

prints 7

Remarks

A null string has zero length.

11.52 LET

11.52.1 Assign a value to a variable

Syntax

LET *variable-name* = *numerical-expression*
LET *string-variable* = *string-expression*

Examples

```
LET C = 5
LET D3 = B
LET J = SIN(X)
LET A$ = "HELLO"
LET A$ = B$ (see notes below)
```

Use of LET in an assignment statement is optional, so the following formats have exactly the same effect:

```
C = 5
D3 = B
J = SIN(X)
A$ = "HELLO"
A$ = B$ (see notes below)
```

Remarks

If a string variable is too short to hold the entire string, it will be truncated without error. For example:

```
10 A$ = "12345678901234567890123456789Othe end"
20 PRINT A$
```

will print 123456789012345678901 (the first 20 characters), because strings default to a maximum length of 20 characters.

Strings can be created by multiple additions. For example:

```
A$ = A$ + MID$(C$,4,3) + CHR$(65)
```

String expressions can also be created from complex string expressions of up to 10 functions.

For example:

A$ = LEFT$(RIGHT$(MID$(A$,3,N),A),B)

The derivation of the function requires the creation of an expression stack which can handle a maximum of ten functions. If this limit is exceeded a 'string complexity' error will result.

11.53 LINCHR

11.53.1 Input a single character from the RS232 port

Syntax

LINCHR [*prompt*] [; or ,] *numeric-variable*

Examples

```
10 LINCHR "DECIMAL VALUE",A
20 PRINT A
30 GOTO 10
```

displays:

```
DECIMAL VALUE 65
DECIMAL VALUE 66
```

etc. as data is read from the port.

Remarks

LINCHR does not wait for carriage return to be received and the character input is not echoed to the screen. A prompt message is displayed on the LCD screen as with INPUT Only one byte will be received. The variable is set to the ASCII code of the character, as detailed in Appendix III.

A comma following the prompt message will suppress the prompt character '?'

11.54 LINE

11.54.1 Draw lines and boxes on the screen

Syntax

Line [(*x1,y1*)] - (*x2,y2*) [,*colour*[,B[F]]]
where *x1*, *y1*, *x2* and *y2* and *colour* are numeric expressions. $0 < x < 239$ and $0 < y < 63$

Examples

10 LINE (10,10) - (50,60)
Draws a line from the co-ordinates (10,10) to (50,60)

10 LINE(0,0)-(239,63),1,BF
Fills the graphics screen with a black box.

Remarks

LINE draws on the graphics screen. If this is not yet selected, BASIC will switch to it before performing the command.

If the first pair of co-ordinates (*x1,y1*) are omitted, the line or box will be drawn from the current position of the graphics cursor

The *colour* parameter denotes the colour of the line. If it is odd then the lines are drawn in black and if it is even then it is drawn in white (ie erased). The colour defaults to black if *colour* is omitted.

The B option draws a vertically aligned box with (x1,y1) and (x2,y2) specifying the co-ordinates of the top left and bottom right corners respectively.

The BF option draws a box and fills it with the specified colour.

The *colour* parameter must be specified if either the B or BF option is used.

11.55 LINPUT

11.55.1 Input a line of data from the RS232 port

Syntax

LINPUT [,][*prompt* [, or ,]] *variables*

Examples

LINPUT A
Inputs a numeric value from the serial port and places it in A.

LINPUT "PARAMETER VALUES";A,B,C
Displays the prompt PARAMETER VALUES? and inputs the values of A, B and C.

Remarks

An input line terminates when a carriage return (CR) character is received. It is then parsed and assigned to the variable list as for an INPUT statement.

LINPUT cannot be used as a direct statement.

The characters input from the serial input port are NOT echoed to the screen.

A comma following the prompt message will suppress LINPUT's '?' prompt.

NOTE: If LINPUT's argument is a simple variable and an invalid entry is made, LINPUT will simply re-prompt without an error message. For example:

10 LINPUT A

If **Hunter 2** receives 'HELLO' and a CR, LINPUT generates a loud 'beep' and re-prompts on the next line.

While the **Hunter 2** is waiting for the first character to be received from the port, it will 'beep' at 30 second intervals.

11.56 LIST

11.56.1 List program on the screen

Syntax

LIST [*first*] [-[*last*]]

Examples

LIST 100-
Lists from line 100 to the end of the program. If there is no line 100 it will LIST from the next line in sequence.

Remarks

By using *first* and *last* you can list any section of the program. LIST without parameters lists the whole program.

11.57 LLIST

11.57.1 List program to the RS232 port

Syntax

LLIST [*first*] [-[*last*]]

Examples

LLIST -500
Lists from the start of the program up to and including line 500.

Remarks

By using *first* and *last* you can list any section of the program. LLIST without parameters lists the whole program.

LLIST can be terminated at any time by holding down ESC until **Hunter 2** responds with a confirmatory 'beep'. Because a transmission buffer is used, a further 256 characters may remain for transmission after this. Powering off will also abort a listing at any time, without loss of stored data or program.

Check **Hunter 2**'s communication parameters (see Part 2) before using LLIST to make sure that the port is set up for compatibility with the device receiving the data. When a printer is being used, remember to check the baud rate and line feed requirements.

Unless the "Echo" transmission parameter is enabled, the listing will not be echoed onto the screen.

NOTE: Because the communications power supply is powered up by LLIST, a single high-to-low character will appear at the start of the listing. Many systems will see this transition as a single, spurious, character. To avoid this problem, power up the interface first with either·

LPRINT " "
or
OUT 132,1

or edit out this character on the host system.

11.58 LLOAD

11.58.1 Load a BASIC program from the RS232 port

Syntax

LLOAD

Examples

LLOAD

Loads BASIC source from the RS232 port into memory, replacing any current program.

Remarks

A 'WAITING' message should be displayed, followed by a 'LOADING' message once lines start being loaded. There is no indication when loading has finished.

The program lines are read as ASCII text, and added to the current program as if you had typed them in direct mode. To replace the current program (rather than add to it, use NEW first.)

Programs which were output using LLIST onto a storage medium (such as a disk or another **Hunter 2**) may be reloaded using this command. LLOAD is terminated by typing (ESC). See 'Using communications', in Part 2, for a detailed description of program loading/unloading options.

Be sure to check **Hunter 2**'s receiving parameters for compatibility with the device transmitting the data.

LLOAD data goes into memory via the BASIC Interpreter input routine and is syntax checked on the way - if errors or spurious data are found in the incoming data, **Hunter 2** will stop loading and display·

* STX Error

If this occurs at the start of a file, it is likely that some header data, not in BASIC syntax, has caused the problem. Simply type LLOAD again, quickly, as **Hunter 2**'s input buffer will still be receiving data.

LLOAD does not echo incoming characters onto the screen.

When you think the program has finished loading, press (ESC) to return to Direct mode. To have **Hunter 2** do this automatically, include the following two direct commands at the end of the file:

```
POKE 63441,0 'clear FFLAG to enable CR/LF echo
POKE 63360,0 'clear IPFLAG to keyboard
```

To run the program automatically when it has finished loading, include a RUN command at the end of the file.

LLOAD executes lines which do not start with a line number. You may take advantage of this by using REM lines without line numbers to comment the source listing on the host; when **Hunter 2** LLOADs the program, it will ignore these comments, saving memory space.

11.59 LN

11.59.1 Return a natural logarithm

Syntax

X = LN(N)

where N is a numeric expression.

Examples

A = LN(8)

generates the logarithm to base e of 8. (2.079441601991)

If S = 9.12 then:

LN(S)

generates the logarithm to base e of 9.12.(2.2104698042074)

Remarks

The argument must be greater than zero. If the argument is negative or equal to zero then a 'Magnitude Error' will occur.

To convert a natural logarithm *l* back to the original number, use EXP(*l*).

11.60 LOAD

11.60.1 Load a BASIC program from the RAM disk

Syntax

LOAD *filename* [,R]

Example

LOAD "HUNTER.HBA"

loads the BASIC file HUNTER.HBA into the workspace ready for execution.

LOAD "HUNTER",R

loads the BASIC file **HUNTER.HBA** into the workspace and commences execution.

LOAD "HUNTER.TXT"

loads the file HUNTER.TXT into the workspace.

If the file specified is ASCII rather than tokenised BASIC, BASIC tokenises it automatically and places this form in the workspace. (This has no effect on the original file.)

Remarks

If you omit the extension from the filename, .HBA is assumed.

11.61 LOC

11.61.1 Return the current position in a sequential file

Syntax

X = LOC(*file-number*)

Example

```
100 MAXFILES = 2
110 OPEN "TEST"FOR INPUT AS #2
120 INPUT#2,A
130 IF EOF(2) = 0 THEN GOTO 120
140 PRINT"End of File"
150 PRINT"No. of records read = ";LOC(2)
160 CLOSE#2
170 END
```

When the end of file has been reached the number of records that have been read will be displayed.

Remarks

A record is 128 bytes long.

LOC can also be used to display the number of records written to a file.

The following program could be used to create the data file TEST, read in the above example:

```
10 OPEN "TEST" FOR OUTPUT AS #1
20 FOR X = 1 TO 100
30 WRITE#1,X
40 NEXT
50 CLOSE#1
60 END
```

LOC can also be used to measure the size of a file.

11.62 LOCATE

11.62.1 Position cursor on the screen

Syntax

LOCATE *x,y*

Character set	Max X	Max Y
Text mode	79	23
Char 0	239	63
Char 1-4	30	5

Example

```
10 LOCATE 0,0
```

Places the cursor in the top left-hand corner

Remarks

LOCATE is most useful for locating the cursor before printing text on the display It operates differently in text mode and graphics mode.

Text Mode

Addressing a location outside the current window will cause the window to move so that the new cursor position is visible.

The maximum values for the parameters are X = 79 and Y = 23, being the size of the virtual screen.

Values greater than the maximum for X will cause wrap-around on the same line. Values greater than the maximum for Y will remain at the bottom of the screen.

Graphics Mode

A SCREEN 1 command will reset the cursor to the origin at (0,0). When character set 0 is selected the LOCATE co-ordinates are graphics co-ordinates. Otherwise, the co-ordinates specified in LOCATE refer to the top left hand pixel of a 7 x 5 character cell, the size of which is equivalent to a CHAR 1 character

11.63 LOG

11.63.1 Generate logarithm to base 10

Syntax

A = LOG(*n*)
where *n* is a positive numeric expression.

Examples

LOG(32)

generates the logarithm to base 10 of 32.

If S = 1.319 then:

LOG(S)

generates the logarithm to base 10 of 1.319.

LOG(-6)

generates a 'Magnitude Error' and program execution will stop.

Remarks

The argument must be greater than zero or a 'Magnitude Error' will occur.

To convert a logarithm *l* back to the original number, use 10^*l*.

11.64 LOPCHR

11.64.1 Output characters to the RS232 port

Syntax

LOPCHR *char-1*, *char-2*, ... *char-n*
where *char-1* etc are numeric expressions in the range 0 to 255.

Examples

LOPCHR 61,62 outputs = >

Remarks

LOPCHR provides a convenient way to output characters by specifying their ASCII codes. This is particularly convenient for control codes.

Also see LPRINT

11.65 LPRINT

11.65.1 Output text to the RS232 port

Syntax

LPRINT [*print-items*][, or ,]
where *print-items* are string or numeric variables, constants or expressions, separated by commas or semi-colons.

Examples

LPRINT A,B,C$

Outputs the values of variables A, B and C$ to the serial output port, followed by a CR.

Remarks

LPRINT is exactly equivalent to PRINT except that output goes to the RS232 port and not the screen.

Values are output as ASCII text, with spaces between each item. See PRINT for a more complete description.

The formatting of output by LPRINT can be altered by using the following:

LPRINT % *n* %	sets the number of decimal places equal to *n*, a number in the range 0-9. Rounding occurs up or down as appropriate.
LPRINT % Z *n* %	sets the number of trailing zeros (or decimal places) equal to *n*, a number in the range 0-9.
LPRINT % E %	formats numbers in scientific notation.
LPRINT % %	restores LPRINT formatting to normal.

Changing LPRINT formatting also affects PRINT formatting.

11.66 LTRON

11.66.1 Trace execution by sending line numbers to RS232 port

Syntax

LTRON

Examples

LTRON

Remarks

LTRON produces the same output as TRON but sends it to the RS232 port, for example to a serial printer

To turn off tracing, use TROFF

See TRON for a full description.

11.67 MAXFILES

11.67.1 Files open limit

Syntax

MAXFILES = *n*
or
n = MAXFILES
where *n* is a numeric variable, expression or constant and is in the range 1< = *n* < = 15

Examples

MAXFILES = 4

Sets the maximum number of files that may be opened simultaneously to four.

Remarks

MAXFILES defaults to 1.

MAXFILES will perform a CLEAR operation destroying the current symbol table (clearing all variables, GOSUB return addresses, DATA pointers and FOR..NEXT counters). It must therefore be used at an early point in the program.

You may also use MAXFILES to determine the current files limit. For example:

100 PRINT "Maximum number of files to be opened = ";MAXFILES

You can also use MAXFILES to reserve space for machine code programs or data (see section 10.11.3).

11.68 MID$

11.68.1 Return or assign a substring

Syntax

B$ = MID$(*string*,*start*,[*length*]) - return a substring
or
MID$(*string-variable*,*start*,[*length*]) = *substring* - assign a substring
where *string* and *substring* are strings and *length* and *start* are numerics.

Examples

10 A$ = "ABCDEFGH"
20 PRINT MID$ (A$,3,4)
Prints CDEF

Remarks

If *length* is omitted then all characters to the right of the specified *start* are returned. In the above example:

MID$ (A$,1,3) is "ABC"
MID$ (A$,20,5) is "" (null string) because 20 is beyond the end of A$
MID$ (A$,7) is "GH"

When MID$ is used to assign a substring, it sets the part of the *string-variable* starting at *start* to the *substring*. The optional parameter *length* denotes how much of *substring* to assign and if omitted, defaults to the length of the *substring*.

11.69 NAME

11.69.1 Rename a file

Syntax

NAME *oldname* AS *newname*
where both parameters are strings

Examples

NAME "VERSION1.HBA" AS "VERSION2.HBA"

Renames the file VERSION1.HBA to VERSION2.HBA

NAME A$ AS B$

Remarks

The NAME command only alters the filename.

Filenames can be up to eight characters long with an optional extension of "." followed by up to three more characters.

11.70 NEW

11.70.1 Clear the current program, variables, etc

Syntax

NEW

Examples

NEW

Remarks

NEW erases all of the current program, so should be used with caution!

Always use NEW before loading a new user program into **Hunter 2** to ensure the maximum use of space.

NEW replies with:

Are you sure? (Y/N)

If any response to the confirmation message other than Y (YES) is made, then the function is aborted and existing programs are not affected.

Following a response of 'Y', NEW displays the number of free bytes available before returning to BASIC.

Files are not affected by NEW

11.71 NEXT

11.71.1 Complete a FOR..NEXT loop

Syntax

NEXT [*variable*]
where *variable* is a variable name that must correspond with the counter variable specified in the matching FOR statement

Examples

NEXT I or NEXT

Remarks

See FOR.

11.72 ON BREAK

11.72.1 Control trapping of the Break key

Syntax

ON BREAK GOTO [*line*] - enable trapping

or

ON BREAK OFF - disable trapping

Examples

ON BREAK GOTO 1000
ON BREAK OFF

Remarks

This statement operates in a similar manner to the ON POWER statement, except that the break key (ESC or CTL/FN C) is monitored instead of the PWR (power) key

11.73 ON COM

11.73.1 Define a subroutine to execute when serial data is received

Syntax

ON COM GOSUB *line-number*

Example

```
10 ON COM GOSUB 100
20 COM ON


100 'SUBROUTINE TO READ
110 'INCOMING DATA

.
190 RETURN
```

Remarks

ON COM is used in conjunction with the COM ON statement. It tells BASIC where to jump to in the event of a communications interrupt.

Data may be left in the communications buffer for as long as is necessary.

On entry to the interrupt-handling routine, an automatic COM STOP is executed, so communications interrupts are never nested. The RETURN from the trap routine performs an automatic COM ON, so re-activating the interrupt routine, unless a COM OFF statement was executed before the RETURN.

If a communications interrupt is handled while **Hunter 2** is in HELP mode, the label line will be lost. To restore the label line, the interrupt-handling routine should reset the HELPFLG location on exit by executing the instruction POKE 63379,0.

11.74 ON COMMS

11.74.1 Control comms error handling

Syntax

ON COMMS GOTO *line* - enable handling by the specified user routine

or

ON COMMS OFF - re-enable handling by the operating system

Examples

```
ON COMMS GOTO 1000
ON COMMS OFF
```

Remarks

The statement operates in a similar manner to ON POWER.

After ON COMMS GOTO, when a communications error is detected (typically a protocol failure), control returns to BASIC which executes the user's error-handling routine, starting at *line*.

To return to normal communications error handling (by the operating system), use ON COMMS OFF.

To analyse the error, read the byte stored at the COMERR location. This is the error code. The codes and their meanings are described in more detail in 'Using communications' in Part 2.

In a typical application, ON COMMS and COMERR would be used to present the operator with a plain language instruction something like this:

"Communications have failed. Please re-dial and try again".

Retries or repeat blocks can also be controlled by the user program, as required.

If a communications error is handled while **Hunter 2** is in HELP mode, the label line will be lost. To restore the label line, the error-handling routine should reset the HELPFLG location on exit by executing the instruction POKE 63379,0.

11.75 ON ERROR

11.75.1 Control trapping of BASIC errors

Syntax

ON ERROR GOTO *line* - turn trapping on
ON ERROR GOTO 0 - turn trapping off

Examples

```
10 ON ERROR GOTO 1000
20 INPUT A
30 A = SQR(A)
40 PRINT A
50 GOTO 20
1000 A =ABS(A)
1010 PRINT "NEGATIVE NUMBER CORRECTED"
1020 RESUME 30
```

This example will trap negative numbers and correct the condition. (Of course, a more elegant solution in this case would be to test A before using it in line 30!)

Remarks

ON ERROR causes program execution to be diverted to the user's error-handling routine if an error is encountered.

Any error can be trapped. However, the error-handling routine will normally only cater for correctable errors such as magnitude errors resulting from incorrect operator input or attempts to open a file which does not exist, etc.

The error-handling routine should end with a RESUME statement (which transfers execution without an error message) or ON ERROR GOTO 0, which terminates execution with the normal BASIC error message for the error

11.76 ON n GOSUB

11.76.1 Call one of a list of subroutines depending on *n*

Syntax

ON *n* GOSUB *line-1*, *line-2*, .. *line-z*
where *n* is a numeric expression or variable in the range 0 to 255.

Examples

```
10 ON A GOSUB 100,180
20 PRINT "HELLO"
```

If A=1 the program branches to line 100. If A=2 the program branches to line 180. In either case, the program returns to line 20 when a RETURN is executed.

```
10 B=100:C=300:D=350
20 ON A GOSUB B,C,D
30 PRINT "HELLO"
```

This example shows variables being used for line numbers.

Remarks

ON GOSUB will cause program execution to go to the *n*th line number in the list after the GOSUB. If *n* has a fractional part, the fractional part is ignored ie *n* is treated as INT(*n*).

If there is no *n*th line number, execution will continue with the next statement, without error

Once the program has branched to one of the line numbers, each statement following that line number will be executed in order until a RETURN is executed. At this point execution will continue with the next statement after the ON GOSUB.

For example, in the above examples, if A=0 then "HELLO" will be printed IMMEDIATELY

11.77 ON n GOTO

11.77.1 Branch to one of a list of lines depending on *n*

Syntax

ON *n* GOTO *line-1*, *line-2*, .. *line-z*
where *n* is a numeric expression or variable in the range 0 to 255

Examples

10 ON A GOTO100,200,300
20 END

If A = 1 then the program will go to line 100. If A = 2 then the program will go to line 200. If A = 3 then the program will go to line 300

Remarks

ON GOTO will cause program execution to go to the *n*th line number in the list after the GOTO. If *n* has a fractional part, the fractional part is ignored ie *n* is treated as INT(*n*).

If there is no *n*th line number, execution will continue with the next statement, without error.

For example, in the above example, if A = 0 or A = 4 then line 20 will be executed next.

11.78 ON KEY

11.78.1 Define a soft key handling routine

Syntax

ON KEY (*key*) GOSUB *line*
where *key* is as follows:

key	meaning
1 to 8	CTL/FN 1 to 8
11	CTL/FN ↑
12	CTL/FN ←
13	CTL/FN →
14	CTL/FN ↓

Examples

ON KEY(1) GOSUB 5000

If function key 1 is pressed the program will execute the subroutine located at line 5000.

Remarks

The *line* parameter specifies the start of the key-handling routine, which must end with a RETURN.

To activate the trap, use KEY (*key*) ON.

11.79 ON POWER

11.79.1 Control trapping of the Power key (PWR)

Syntax

ON POWER GOTO *line* - turn trap on

or

ON POWER OFF - turn trap off

Examples

ON POWER GOTO 1000

Remarks

ON POWER loads a vector into the OFFVECT locations so that if PWR is pressed, program execution occurs from the line number specified. PWR will not power the **Hunter 2** off.

Operation of ESC, detection of a syntax error, or execution of ON POWER OFF will cause the vector to be re-initialised to enable PWR to function normally again.

11.80 ON POWER RESUME

11.80.1 Put Hunter 2 into CONT mode

Syntax

ON POWER RESUME

Examples

ON POWER RESUME

Remarks

After **Hunter 2** has been switched off, by pressing PWR or any other means, when you switch on again, the program will continue from the point where it was switched off, even if input was in progress.

To exit from CONT mode hold down CTL/FN and C during power-up, then release these keys and type the 'escape code'

In CONT mode, you must also type the 'escape code' after using the Break key (CTL/FN C or ESC), to stop program execution.

When ON POWER RESUME is executed, the keyboard input buffer is cleared of any pending keyboard inputs and the keyboard is reset so that no keyboard activity is recognised for another 18 milliseconds. This means that you should not execute ON POWER RESUME in very short loops, as all keyboard input will then be ignored.

11.81 ON TIME$

11.81.1 Define a routine to execute at a specified time

Syntax

ON TIME$ = *time* GOSUB *line*
where *time* is a string defining the time at which to execute the subroutine, which starts at *line*. The string must be in the format "*h*[:*m*[:*s*]]", where *h*, *m* and *s* are one or two digits specifying the time, using the 24-hour clock.

Example

10 ON TIME$ = "12:00:00" GOSUB 1000

500 ON TIME$ = "13" GOSUB 2010

Remarks

The time is checked before executing each statement, and if it matches the specified time, the routine is called (see GOSUB).

This means that if BASIC is waiting for an input when the time is reached, the routine will not be executed until the input is received, since that is when the next statement is executed.

If omitted, *m* and *s* default to 00.

ON TIME$ has no effect while **Hunter 2** is powered down.

11.82 OPCHR

11.82.1 Output characters to the screen

Syntax

OPCHR *char-1*, *char-2*, etc
where *char-1* etc are numerics giving the ASCII codes of the characters; 0 < = *char* < = 255

Examples

OPCHR 61,62 displays = >
OPCHR 1 clears the screen

Remarks

OPCHR allows you to specify characters by their ASCII codes which is more convenient when you need to output control codes or use advanced functions such as cursor addressing.

For example, cursor addressing uses ASCII code 15:

OPCHR 15,*x*,*y*

This will move the cursor to the screen co-ordinates specified by the next two characters output (*x* and *y*). Note that re-positioning the cursor in this manner will not move the virtual screen window over the virtual screen automatically. If this is required, use LOCATE.

11.83 OPEN

11.83.1 Open a file for sequential input/output

Syntax

OPEN *filename* FOR *mode* AS *file-number*
where *mode* is OUTPUT, APPEND or INPUT

Examples

10 OPEN "OUTFIL.TXT" FOR OUTPUT AS 1

This will open the file "OUTFIL.TXT" for output from a BASIC program as file number one, to allow the BASIC program to write data to it. Any existing file of that name will be replaced.

10 OPEN "APPEND.TXT" FOR APPEND AS 3

This opens the file "APPEND.TXT" as file number three. Any data written to the file will be added to its end. (MAXFILES must have previously been set to a minimum of three.)

10 OPEN "INPUT TXT" FOR INPUT AS 4

The file is opened for input as file number four, to allow the data in the file to be read by a BASIC program.

Remarks

When a file is OPENed in INPUT or APPEND mode, **Hunter 2** generates a checksum for it and compares it to the checksum stored in the file directory, when the file was last CLOSEd. Any inconsistency indicates that the file has become corrupted and will result in a 'DSK ERROR' being returned to the user

The OPEN statement has three options: OUTPUT, APPEND and INPUT. These operate as follows:

OUTPUT This will open a file for output. Any data that is in the file will be lost.

APPEND This will also open a file for output, but any data that is output to the file will be added to the end of the data that is already present.

INPUT The file is opened for input. This will allow a BASIC program to read data from the file.

The file number must be no greater than the number specified in MAXFILES. If MAXFILES 3 has been specified the file number must be no greater than 3.

11.84 OUT

11.84.1 Output a byte to a port

Syntax

OUT *port*, *byte-1*, *byte-2*, etc

Examples

OUT 133,1

Outputs a 01 byte to port 133 (which switches DTR off).

Remarks

OUT can be used to control internal **Hunter 2** functions like power control. See Part 4 for more details of ports.

11.85 PEEK

11.85.1 Read a byte from memory

Syntax

X = PEEK(*address*)
where *address* is a numeric expression in the range 0 to 65535

Examples

PRINT PEEK (17933)

displays the value of location 17933, in decimal.

Remarks

PEEK requires the decimal value of the required memory location as an argument. See also DEFSEG to set the page address.

See Part 4 for a description of the useful addresses in **Hunter 2**'s memory map.

11.86 PI

11.86.1 Return the value of PI

Syntax

X = PI

Example

```
10 INPUT "ANGLE IN DEGREES",A
20 LET B = A*PI/180
30 PRINT A;"DEGREES ARE";B;"RADIANS"
```

This program will convert an angle in degrees to an angle in radians.

Remarks

PI returns a value accurate to 5 decimals places: 3.14159. For more accurate mathematical functions use a double precision variable to store a more precise value for PI. For example:

PI!(0) = 3.1415926535896

11.87 POINT

11.87.1 Return the colour of a screen pixel

Syntax

A = POINT (*x*,*y*)
where *x* is a numeric in the range 0 to 239 and *y* is a numeric in the range 0 to 63

Example

```
10 REM Invert the graphics screen
20 FOR X = 0 TO 239
30 FOR Y = 0 TO 63
40 IF POINT(X,Y) = 1 THEN PRESET (X,Y) ELSE PSET (X,Y)
50 NEXT Y.NEXT X
```

Remarks

If POINT returns an odd number, the pixel is black. If POINT returns an even number, the pixel is white. If the co-ordinates are out of range then POINT returns -1, without error.

11.88 POKE

11.88.1 Write bytes to memory

Syntax

POKE *address*, *byte-1*, *byte-2*, .. *byte-z*
where *address* is a numeric in the range 0 to 65535 and *byte-1* etc are numerics in the range 0 to 255

Examples

POKE AD,1

Pokes the value 1 into the address given by AD.

Remarks

If a number of bytes are specified in the POKE command, they will be written to consecutive addresses, starting at *address*

See also DEFSEG to set the page address.

WARNING: Use POKE with care.

See Part 4 for a description of the useful addresses in **Hunter 2**'s memory map.

11.89 POP

11.89.1 Fetch values from the machine code stack

Syntax

X=POP(*n*)
where *n* is the number of 16-bit values to POP off the stack.

Examples

A=POP(2)

POPs two values off the stack, assigning the second to the variable A

Remarks

POP allows you to access 16 bit values put on the stack by machine code subroutines.

When the argument is more than 1, POP skips past values on the stack. You will then be unable to read these values.

Refer to section 10.11 for details of using machine code calls.

WARNING: Do not attempt to pop more variables from the stack than have been pushed.

11.90 POS

11.90.1 Return the horizontal position of the cursor

Syntax

X = POS(*n*)
where *n* is any numeric (for example, 0).

Examples

A = POS(0)

Remarks

POS returns a value in the range 1 to 80.

11.91 POWER

11.91.1 Set auto power-off period

Syntax

POWER 0 - disable auto power-off
or
POWER *n* - set power-off period
where *n* is in the range 10-255, specifying the number of 5 second periods.

Examples

POWER 12
will switch **Hunter 2** off after 1 min.

POWER 0
will prevent timed switch off.

Remarks

If the **Hunter 2** is left waiting for an input, it will remind the user that it has been left switched on, by emitting a beep every 150 seconds. If no input is received within the auto power-off period, it will power off automatically.

POWER 0 disables auto power-off and re-enables PWR after using POWER CONT

The power-off period is reset to the default value (n = 60, or 5 minutes) by powering off and on again, so it should be re-initialised by the user program.

11.92 POWER CONT

11.92.1 Disable the power key and power off time-out

Syntax

POWER CONT

Examples

POWER CONT

Remarks

Use of POWER CONT allows the user program to control power off, for instance, to make sure that data entry is complete before allowing the **Hunter 2** to power down.

To re-enable [PWR] and the power off time-out, use POWER *n*.

11.93 POWER OFF

11.93.1 Switch off Hunter 2

Syntax

POWER OFF

Examples

POWER OFF

Remarks

After POWER OFF has been executed switching **Hunter 2** on will automatically run the program from the start.

To exit from POWER OFF, press [ESC] and type the emergency breakout code.

POWER OFF can be used in applications where CONT *filename* could give confusing results if the unit is left in an uncontrolled state by a previous user.

11.94 POWER OFF, RESUME

11.94.1 Switch off Hunter 2 with automatic continuation

Syntax

POWER OFF, RESUME

Examples

POWER OFF, RESUME

Remarks

POWER OFF, RESUME turns the **Hunter 2** off. When it is turned on again, the screen will be restored and execution will re-start from the next statement in the program.

To break out of the program, simply press ESC. There is no need to type the emergency breakout code, unlike using POWER OFF

11.95 PRINT

11.95.1 Display data on the screen

Syntax

PRINT [@(*x*,*y*)] *arguments* [;]

or

? [@(*x*,*y*)] *arguments* [,]

where *x* and *y* are numeric expressions specifying the coordinates at which to print and *arguments* is a list of numeric or string expressions to print starting at the current or specified position. The arguments must be separated by commas or semi-colons

Examples

PRINT X,Y,Z$

Prints the values of the three variables X,Y, and Z$ in columns 1, 14, and 28 and finishes with CR/LF

PRINT @(0,0)"Hello ";n$

Prints a greeting at the top right-hand corner of the screen.

PRINT

Moves the cursor to the start of the next line.

Remarks

The comma between the expressions move the print cursor to the next zone, where each zone is fourteen spaces wide. The use of semi-colons causes the values to be printed together without gaps. However, printing numerical data leaves a space at the start for the minus sign, and includes one blank space afterwards. A semi-colon at the end of the line suppresses the CR/LF.

Examples:

PRINT "FOOT" + "BALL"
outputs: FOOTBALL

PRINT X;"SQUARED IS";
PRINT X*X
If X is 2, outputs: 2 SQUARED IS 4

IF A<2 THEN PRINT "A=";A
If A is 1, outputs: A=1

PRINT CHR$(1)
Clears the LCD display.

PRINT formats can be altered by the following statements:

PRINT % *n* %	sets the number of decimal places to *n*, a number in the range 0-9. Rounding occurs up or down as appropriate.
PRINT % Z *n* %	sets the number of trailing zeros (or decimal places) to *n*, a number in the range 0-9
PRINT % E %	sets PRINT formatting to scientific notation.
PRINT % %	restores PRINT formatting to normal.

Changes to PRINT formatting also affect LPRINT formatting.

11.96 PRINT#

11.96.1 Write data to a file

Syntax

PRINT *#file-no, arguments* [;]
where *file-no* identifies a file which is open for output or append and *arguments* is a list of numeric or string expressions, separated by commas or semi-colons

Examples

PRINT#4,A$;B4;D$
PRINT#1,X,Y;Z

The variables specified are written to the specified file.

Remarks

PRINT# formats the output to the file exactly as PRINT formats output to the screen, including the PRINT format set by PRINT %E% etc (see PRINT).

The arguments should usually be separated by semi-colons and not commas, to minimise the amount of file space wasted by writing spaces.

If string data is written to the file, when it is read back any commas, semi-colons, leading blanks, carriage returns or line will be interpreted as delimiting characters. If you want to be able to read delimiting characters, write a double quote character before and after the string containing them, for example by using CHR$(34).

For example, if A$ ="TOM,DICK" and B$="HARRY" The statement PRINT#2,A$;B$ would write the following to the file:

TOM,DICKHARRY

When it was read back with a statement like this:

INPUT#2,A$,B$

'TOM' would be read into A$ and 'DICKHARRY' into B$.

To separate the data correctly the following format should be used:

PRINT#2,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)

This would write the following to the file:

"TOM,DICK""HARRY"
and the statement:
INPUT#2,A$,B$
would input 'TOM,DICK' to A$ and 'HARRY' to B$
WRITE#n provides a simpler way to achieve the same results.

11.97 PSET and PRESET

11.97.1 Set or reset screen pixels

Syntax

PSET (*x1,y1*),(*x2,y2*),.. - set pixels to black

PRESET (*x1,y1*),(*x2,y2*),.. - set pixels to white

(*x1,y1*) etc specifies the coordinates of the pixel to be set or reset and must lie in the range $0 <= x <= 239$ and $0 <= y <= 63$. The coordinate (0,0) refers to the top left corner of the screen.

Examples

PSET(0,0),(239,0),(0,63),(239,63)

Will set the four corner pixels to black.

Remarks

PSET and PRESET are graphic statements, so they will switch the screen to graphics mode if not already in it.

The graphics cursor is left on the last pixel specified in the PSET or PRESET statement.

11.98 PUSH

11.98.1 Push values onto the machine code stack

Syntax

PUSH *n-1*, *n-2*, *..n-z*
where *n-1* etc are numeric and in the range 0 to 65535

Examples

PUSH A,B

pushes the integer (16 bit binary) representation of the variables A and B onto the stack.

Remarks

This data may then be accessed or modified by a machine code program. See section 10.11.

POP is the reverse of PUSH.

11.99 READ

11.99.1 Read constants or variables from a DATA statement

Syntax

READ *variable-1*, *variable-2*, *..variable-z*
where *variable-1* etc are numeric or string variables

Examples

READ X
sets X equal to the current DATA constant.

For example the program:

```
10 DIM D!(10):DIM A$(10,10)
20 DATA 3,4,5,2/3,"HELLO"
30 READ A,B,C,D!(0),A$(1)
40 PRINT A,B;C;D!(0);A$(1)
```

Prints:

3 4 5 .66666666666667 HELLO

with A, B and C equal to three sequential DATA constants, D!(0) a double precision constant and A$(1) a string expression.

Remarks

The constants in DATA statements in the program are considered as a single list, in line number order READ reads sequentially through the list until it is exhausted, when an error will result. The current position in this list may be changed to the start of any DATA statement by using RESTORE.

The variable type specified in a READ command must match the DATA item being read: if not, a READ ERROR will occur

11.100 REM

11.100.1 Ignore remainder of line

Syntax

REM *comment*

or

' *comment*

Example

```
1000 REM THIS ROUTINE CLEARS THE DISPLAY
1010 REM
1020 PRINT CHR$(1);
1030 RETURN
```

Remarks

REM statements are ignored by the program. They can considerably improve readability but only at the expense of extra memory. REM statements without line numbers can be used in host computer files which will be ignored when down loaded.

Use an apostrophe (') instead of :REM on a multi-statement line to save memory space:

```
10 X=X+1' THIS TYPE OF COMMENT SAVES MEMORY SPACE
```

REM statements are also used to define HELP text.

11.101 RESTORE

11.101.1 Move data pointer used by READ

Syntax

RESTORE [*line-number*]
where *line-number* is a constant in the range 0 to 65535.

Example

```
RESTORE 120
```

This restores the data pointer to line 120.

Remarks

Used without a *line-number*, RESTORE moves the READ pointer to the first DATA statement in the program.

11.102 RESUME

11.102.1 Return from an error-handling routine

Syntax

RESUME [*line*]
where *line* is either a line number (constant, variable or expression) or NEXT

Example

```
10 ON ERROR GOTO 100
20 IF A<B THEN ERROR 110


100 IF ERR>100 THEN RESUME L2+(ERR/10)
```

Remarks

RESUME can only be used to leave an error-handling routine, entered via an ON ERROR GOTO statement.

If there is no argument RESUME will resume execution with the statement that caused the error

RESUME NEXT resumes execution with the statement after the one that caused the error

RESUME *line* resumes execution with the specified line.

The only other way to leave an error-handling routine is by the statement ON ERROR GOTO 0, which allows BASIC to handle the error in its usual way (with an error message and program termination).

11.103 RETURN

11.103.1 Return from a BASIC subroutine

Syntax

RETURN

Examples

```
10 GOSUB 100


100 PRINT I
200 RETURN
```

Remarks

RETURN is the logical conclusion of a GOSUB statement. Placed at the end of the subroutine, RETURN causes execution to continue at the next statement following the original GOSUB statement.

A subroutine may contain more than one RETURN.

11.104 RIGHT$

11.104.1 Return the last n characters of a string

Syntax

A$ = RIGHT$(*string*,*n*)
where *n* is a numeric giving the number of rightmost characters to be returned and *string* is a string constant, variable or expression.

Examples

PRINT RIGHT$("ABCDEF",3)
prints DEF

Remarks

If the string is shorter than *n* then the entire string is returned, without error. For example:

A$ = "ABCDEF"
RIGHT$(A$,0) returns a null string
RIGHT$(A$,20) returns ABCDEF

11.105 RND

11.105.1 Return a random number

Syntax

X=RND(*n*)
where *n* is a dummy numeric argument, usually 0.

Examples

A = RND(0)
sets A equal to a random number

Remarks

RND returns a 14 digit pseudo-random number in the range 0-0.99999999999999

11.106 RUN

11.106.1 Run a BASIC program

Syntax

RUN [*line*] [*filename*]
where *line* is a numeric constant, variable or expression stating the line number to start from and *filename* is a string constant, variable or expression specifying the program to load and run

Examples

RUN 1000
executes from line 1000.

RUN A*100
executes from line A*100.

RUN "filename"
loads "filename" into the BASIC memory and executes it.

Remarks

RUN without a *filename* executes the BASIC program that is in memory. RUN without a *line* executes the program from its first line.

RUN is useful when debugging programs as it initialises the system stacks, unlike GOTO.

RUN may be used within a program to load and run another program.

If RUN loads an ASCII file, it will convert it to tokenised form automatically before running it. (This has no effect on the original file.)

11.107 SAVE

11.107.1 Save current program as a file

Syntax

SAVE *filename* - save as a tokenised file
or
SAVE *filename*, A - save as an ASCII file
where *filename* is a string constant, variable or expression

Example

SAVE "HUNTER.HBA"

Remarks

The current BASIC program is written to the specified file. If a file already exists with that name, it is overwritten. Filenames are up to eight characters long. If an extension is omitted, .HBA is used.

If there is insufficient file space for the file, an appropriate error message will be displayed.

11.108 SCREEN

11.108.1 Select text or graphics screen

Syntax

SCREEN *n*
where *n* is a numeric constant, variable or expression returning a value of 0 or 1. 0 puts the screen into text mode, 1 into graphics mode.

Example

SCREEN 0
puts the screen into text mode.

Remarks

TEXT MODE (SCREEN 0)

This is the main mode of operation, used to list BASIC programs and to enter programs. In this mode the output is purely textual and the screen gives a window of 8 lines of 40 characters on a virtual screen of 24 lines of 80 characters.

Switching to graphics mode and back again preserves the contents of the text screen.

GRAPHIC MODE (SCREEN 1)

In this mode the user is given much more control over the screen with high resolution graphics and up to 5 different character sizes.

Switching to text mode and back again erases the contents of the graphics screen.

Graphic commands select graphics mode automatically

11.109 SGN

11.109.1 Return a value indicating the sign of the argument

Syntax

A=SGN(*n*)
where *n* is a numeric constant, variable or expression.

Examples

A = SGN(-9.5)
sets A equal to -1

A = SGN(0)
sets A equal to 0

A = SGN(1.76E9)
sets A equal to 1

Remarks

SGN(*n*) returns results as follows:

n	Result
n<0	-1
n=0	0
n>0	1

11.110 SIN

11.110.1 Return Sine of an angle

Syntax

A = SIN(*angle*)
where *angle* is a numeric constant, variable or expression in radians.

Examples

J = SIN(2.56)
sets J equal to the value of the sine of 2.56 radians.

PRINT SIN(2.56)
displays: .549355436427

Remarks

If the angle is in degrees, you must convert it to radians, by multiplying by PI and dividing by 180. For example:

J = SIN(A*PI/180)

SIN is accurate to 12 decimal places.

11.111 SOUND

11.111.1 Generate a tone

Syntax

SOUND *pitch,duration*
where *pitch* and *duration* are numeric constants, variables or expressions, in the range 1-65534.

Example

SOUND 121,530
will produce a note which is approximately middle 'C', for one second.

The following program will play a short tune:

```
10 FOR I=1 TO 20
20 SOUND 52,1233/I
30 SOUND 46,1394/I
40 SOUND 59,1086/I
50 SOUND 121,530/I
60 SOUND 80,1202/I
70 FOR J=I TO 16
80 NEXT J
90 NEXT I
99 END
```

Remark

The higher the *pitch*, the lower the note. The higher the *duration*, the longer the note.

To play a note for 1 second, use a *duration* of (64130/*pitch*).

See Appendix VIII for the parameters required to generate notes of a desired frequency

11.112 SPACE$

11.112.1 Return a string of spaces

Syntax

A$=SPACE$(*length*)
where *length* is a numeric constant, variable or expression in the range 0 to 255.

Example

```
10 DIM A$ (0,10)
20 A$ = "123" + SPACE$(3) + "ABCD"
30 PRINT A$
```

This will print a string consisting of "123" followed by 3 spaces and "ABCD"

Remarks

SPACE$ provides an easier and more compact way of specifying blank strings than using a constant. Also see STRING$.

11.113 SPC

11.113.1 Print spaces in a PRINT statement

Syntax

PRINT SPC(*spaces*)
where *spaces* is a numeric constant, variable or expression in the range 0 to 255.

Example

```
100 PRINT"HELLO";SPC(10);"HOW ARE YOU ?"
```

Will display:

```
HELLO          HOW ARE YOU ?
```

Remarks

SPC must be used in a PRINT statement.

If used as the last item in a PRINT statement, SPC suppresses the usual CR/LF at the end of the line.

11.114 SQR

11.114.1 Return square root

Syntax

A = SQR(*n*)
where *n* is a positive numeric constant, variable or expression.

Examples

```
Q = SQR(16)
```
sets Q to 4.

Remarks

To find other roots, use logarithms. For example, using natural logarithms:

```
10 INPUT "Number to find root of";N
20 INPUT "Root";R
30 N = LN(N)
40 N = N/R
50 PRINT "Root is:";EXP(N)
```


11.115 SRCH

11.115.1 Search an array

Syntax

A = SRCH(*search-element*,[*n*])
where *search-element* is an element of the array holding the value to search for and *n* is a numeric constant, variable or expression specifying that the search is for the *n*th match.

Examples

```
PRINT SRCH(A$)
A = SRCH(A(5),3)
IF SRCH(A!(0),1) < > 5 THEN OPCHR7
```

Remarks

SRCH can be used with string, simple and double precision arrays.

SRCH searches from element 0 in the array specified by *search-element*, for elements that match the value of *search-element*. It does not consider *search-element* to be a match for itself, so if there are no other elements that match it, no match will be reported.

If a match is found, that element's number will be returned, otherwise 0 will be returned.

WARNING: if *search-element* is not element 0 and element 0 is a match, a value of 0 will be returned, which is indistinguishable from the result for no match.

If *n* is specified, the element number for the *n*th matching element will be returned, or 0 if there is no *n*th match. If *n* is not specified, it defaults to 1 ie the first match will be returned.

Consider the following program:

```
10 DIM A$(10,20)
20 A$(0) = "ABCDE"
30 A$(1) = "AB"
40 A$(2) = "ABCDE"
50 A$(3) = "ABCDE"
60 A$(4) = "ZXYZ"
70 A$(5) = "ABCDE"
```

```
100 INPUT X,Y
110 PRINT SRCH(A$(X),Y)
120 GOTO100
```

The following are the results of search on the above array:

SRCH(A$) returns 2	as element (2) is the first occurrence of element (0).
SRCH(A$,2) returns 3	as element (3) is the second occurrence of element (0).
SRCH(A$(2),1) returns 3	as element (3) is the first occurrence of element (2).
SRCH(A$(4)) returns 0	as element (4) is not found elsewhere.

Matching

When SRCH is used with a numeric array, the compared elements must be exactly equal to match.

When SRCH is used with a string array, the comparison is as described for the IF statement, ie the elements match if the *search-element* string is equal to, or is included as the start of, the element being compared. For example:

```
PRINT SRCH(A$(1))
```

would return 2 as A$(1) is a subset of A$(2).

11.116 STOP

11.116.1 Stop execution with a message

Syntax

STOP

Examples

```
550 STOP
```

stops execution and displays the message:

```
* BREAK IN LINE 550 *
```

Remarks

STOP is mainly used when debugging a program, to allow you to halt the program at crucial stages and examine or alter variables.

Type CONT to continue from the line following STOP.

NOTE: If STOP is used on a multi-statement line, CONT does not continue with the next statement, but with the next line.

STOP may also be used to terminate execution in error situations that the program cannot handle, although this is less satisfactory.

11.117 STR$

11.117.1 Return a string representation of a numeric

Syntax

A$ = STR$(*n*)

where *n* is a numeric constant, variable or expression.

Examples

```
10 A = 123.6
20 B$ = STR$(A)
```

Sets B$ to "123.6"

```
10 A = 7.941E12
20 B$ = STR$(A)
```

Sets B$ to "7.941E12"

Remarks

STR$ allows you to use string operators and functions with numbers. It is particularly useful when formatting numeric information for use in tables, or file output, etc.

NOTE: The string returned by STR$(*n*) is not the same as the text displayed by PRINT *n*. For example, STR$ does not include a leading space if the number is positive.

Exponential notation numbers are represented literally in string form.

11.118 STRING$

11.118.1 Return a string of a specified character

Syntax

A$ = STRING$(*length*,*code*)

or

A$ = STRING$(*length*,*string*)

where *length* and *code are* numerics in the range 0 to 255 and *string* is a string constant, variable or expression.

The first version returns a string of *length* characters with ASCII code *code*. The second version returns a string *length* repetitions of the first character of *string*.

Examples

```
100 PRINT"PAGE HEADING"
110 PRINT STRING$(12,"-")
```

Will output:

```
PAGE HEADING
------------
```

Remarks

The following statements have exactly the same effect:

A$ = STRING$ (5,65)

A$ = STRING$ (5,"A")

each assigning a string of five "A" characters to A$.

11.119 SWAP

11.119.1 Exchange the values of two variables

Syntax

SWAP *variable-1, variable-2*
where the variables are both numeric or both string.

Example

SWAP A,B
will put the value of variable A into B and the value of B into A.

SWAP A$,B$
will put the contents of A$ into B$ and the contents of B$ into A$.

Remark

If you attempt to swap a string variable with a numeric variable, a 'Type Mismatch' error will occur

11.120 SYSTEM

11.120.1 Return to File Manager

Syntax

SYSTEM

Example

100 SYSTEM

Remarks

SYSTEM returns to the File Manager screen, leaving the BASIC program and variables intact. If you select BASIC again, you can continue the program by using CONT

11.121 TAB

11.121.1 Move the cursor to a specified column

Syntax

PRINT TAB(*n*)

where *n* is a numeric giving the position to begin printing, in the range 0 to 255.

Examples

```
100 PRINT"COL 1";TAB(10);"COL 2";TAB(20);"COL 3"
```

Will print:

```
COL 1    COL 2     COL3
```

Remarks

If *n* is less than (ie to the left of) the current print position, TAB is ignored. If *n* is greater than the screen width (79 in text mode and graphics character set 0), TAB will move down the screen too. For example, if *n* is 150, TAB will move down one line and to position (150-80) ie 70 on that line.

TAB may also be used in an LPRINT statement, with similar results. The print position is moved to by printing spaces. The line length is assumed to be 255 for LPRINT output.

11.122 TAN

11.122.1 Return tangent of an angle

Syntax

A=TAN(*angle*)

where *angle* is a numeric constant, variable or expression, in radians.

Examples

```
T = TAN(2.56)
```

sets T to the tangent of 2.56 radians.

```
PRINT TAN(2.56) displays:
-.65744712167262
```

Remarks

The argument is expressed in radians.

The tangents of an angle expressed in degrees may be calculated as follows:

```
A = TAN(angle*PI/180)
```

TAN is accurate to 12 decimal places.

11.123 TIME$

11.123.1 Return or set the time

Syntax

A$ = TIME$ - return the time

or

TIME$ = *time* - set the time
where *time* is a string constant, variable or expression in the form [*hh*[:*mm*]].

Examples

```
10 DIM A$(0,8)
20 PRINT TIME$
30 A$ = "10:5"
40 TIME$ = A$
```

Remarks

TIME$ returns an eight character string in the format *hh:mm:ss*

When setting the time, you may omit leading zeros but, for example, if the minutes are zero, then one zero has to be entered. If you omit the *mm* part, it defaults to 0. Do not attempt to enter the seconds, as that will cause a Syntax error. Seconds are set to "00" when setting the clock.

11.124 TRON and TROFF

11.124.1 Control program tracing

Syntax

TRON - turn tracing on

TROFF - turn tracing off

Example

```
TRON
RUN
Will display:
[10] [20] [30].....
Until a TROFF command is executed.
```

Remarks

Line numbers are displayed in square brackets, but apart from this the program runs normally. This allows you to follow the path of program execution.

TROFF turns off the trace facility.

11.125 VAL

11.125.1 Return numeric version of a string

Syntax

A = VAL(*string*)
where *string* is a string constant, variable or expression which is a decimal number.

Examples

```
10 A$ = "123.6"
20 B = VAL(A$)
```

sets B to 123.6

Remarks

VAL performs the opposite function to STR$.

The *string* must be a valid number, in normal decimal or scientific notation. For example:

```
10 A$ = "2.91E6"
20 B = VAL(A$)
```

sets B to 2910000

11.126 VARPTR

11.126.1 Return address of a variable

Syntax

A = VARPTR(*variable*)

Examples

```
100 PRINT VARPTR(B)
```

This will display the address of variable B.

Remarks

VARPTR can, for example, be used to pass a string variable to a machine code routine, as follows:

```
10 DIM A$(0,100)
20 INPUT "String to pass";A$
30 PUSH VARPTR(A$)
40 CALL (C)
```

See section 10.5 for details of variable storage and the meaning of the value returned by VARPTR.

11.127 WAND

11.127.1 Select barcode protocol

Syntax

WAND = *n*
where *n* is a numeric constant, variable or expression.

Example

WAND = 0

selects the CODE 39 decoder software.

Remarks

Hunter 2 can have multiple barcode decoder packages loaded. This command selects which to use.

Currently implemented are:

0 - Code 39
1 - EAN 8/13
2 - Interleaved 2-of-5

It is essential to have the relevant decoder software loaded in the factory prior to attempting to use a barcode wand. See 'Accessories' in Part 2 for further information.

Setting *n* to 3 will allow use of the monitor package, if loaded in 9G08.

11.128 WHILE..WEND

11.128.1 Execute statements while a condition is true

Syntax

WHILE *logical-expression*

WEND
where *logical-expression* evaluates to true or false. If true, the statements in the loop are executed. If false, execution continues with the statement after WEND. WHILE loops may be nested; each WEND is associated with the most recent WHILE.

Examples

```
10 FOR Q=1 TO 10
20 B(Q)=Q
30 NEXT
110 WHILE A< 11
120 FOR I = 1 TO 10
130 PRINT B(I);
140 NEXT
150 PRINT A.A = A+1
160 WEND
```

The array B is printed as long as the A parameter is less than 11.

Remarks

The *logical-expression* is re-evaluated each time the WEND statement is executed. It is only at this stage that the WHILE loop will exit.

To leave a WHILE loop, change the value(s) tested in the *logical-expression* to make sure that the result is false and jump to the WEND statement. If the WHILE loop is in a subroutine, you may also use RETURN to leave the WHILE loop and subroutine.

Do not jump out of a WHILE loop or a control stack error will result.

WHILE cannot be used with a string expression. The statements WHILE and WEND must be on separate lines and no other statements must be on those lines.

11.129 WINCHR

11.129.1 Input a single character from a barcode wand

Syntax

WINCHR *numeric-variable*

Examples

```
100 WINCHR N
110 PRINT N
```

Reads a single character from the barcode wand as an ASCII code, stores it in N and displays it.

Remarks

This function acts exactly like INCHR, but takes input from the optional barcode wand. The value is returned from the wand exactly like a keyboard input.

If a multiple character barcode is scanned then WINCHR will return the first character and the rest of the code will be ignored.

As with WINPUT if any key is pressed prior to using the wand, then the input is taken from the keyboard instead.

NOTE: The barcode format must first be selected using WAND. The relevant decoder software must be resident in **Hunter 2** as a system file. If it is not, a system file error will result.

11.130 WINPUT

11.130.1 Input a line from a barcode wand

Syntax

WINPUT [,] [*prompt*;] *variable*

See INPUT for more details.

Examples

```
100 WINPUT A$
110 PRINT A$
```

Reads the barcode wand input, stores it in A$ and displays it.

Remarks

The string returned from the barcode is read exactly like a keyboard input. If any key is pressed prior to barcode input then the input will be taken from the keyboard instead.

Further details of available barcodes may be found in 'Accessories' in Part 2.

The use of string variables is recommended to allow for any non-numeric content.

NOTE: The barcode format must first be selected using WAND. The relevant wand software must be resident in **Hunter 2** as a system file. If it is not, then a system file error will result.

11.131 WRITE#

11.131.1 Output data to a file

Syntax

WRITE# *file-number*, *var-1*, *var-2*, ..*var-z*

Examples

```
WRITE#2,A$,B$,C
```

writes the variables A$, B$ and C to file number 2.

Remarks

WRITE# and PRINT# both write data to a file. The output produced by WRITE# differs in three respects:

- items are separated by commas if written by the same WRITE# statement
- strings are surrounded by quotation marks (")
- positive numbers are written without a leading space

HUSKY HUNTER 2™

Part 4
Assembly Language

Chapter 12 Developing Software for the Hunter 2

12.1 Introduction

Many users of the **Hunter 2** will wish to develop their own software to suit their particular needs. Often, they will wish to use languages and computers with which they are familiar for the development process, with the final result being a program which will run on the **Hunter 2**.

This chapter describes several tried and tested program development methods. It is by no means an exhaustive list and should not be taken as a recommendation or otherwise, of a particular product.

For details of using **Hunter** BASIC, refer to Part 3.

12.2 Interpreted Microsoft BASIC

MBASIC is the Microsoft BASIC Interpreter. Programs developed with it can be run under MBASIC or compiled using BASCOM and run as stand-alone applications.

MBASIC requires no special configuration to run on the **Hunter 2**. It is simply loaded by transferring the file MBASIC.COM. When MBASIC is run, the Interpreter and editor are loaded into TPA, reducing by 24K the space available for the program.

This need not be a restriction on program or data size, because MBASIC allows chaining of programs. Each program is able to call another program, (resident on **Hunter 2**'s RAM disk), which then replaces it in the TPA and runs. Variables common to the two programs can be declared by statements in the first program, that is the one which initiates the chain function. Since a program is lost from TPA when it chains another, all programs must reside on the RAM disk.

A program developed using MBASIC on a desktop CP/M machine will usually run successfully on a **Hunter 2** without modification. If the desktop machine uses a different operating system, there is still a good chance that an MBASIC program can be transferred to and run on **Hunter 2**, providing that no features specific to the operating system are employed. The core of the language and its functions are common between versions of MBASIC.

12.3 Compiled Microsoft BASIC

Compiled Microsoft BASIC offers the advantages of execution speed and program security compared with interpreted BASIC. However, these must be set against several disadvantages.

The compiled code must be resident in the TPA for execution, along with its run-time system and libraries. In the case of Microsoft BASIC, this reduces the TPA available for program and data by about 16K. This often means that the program will not fit in TPA in one piece so it must be split into overlay segments which will all reside on the RAM disk. So, while a **Hunter** BASIC program with data may just fit into the TPA leaving the whole of the RAM disk free, the equivalent compiled version may occupy 50K or more of RAM disk. This obviously affects the size of memory required on the **Hunter 2**.

BASIC compilers do not incorporate the special features available in **Hunter** BASIC, such as those relating to communications and barcode wand input. However, these can be provided by means of machine code calls.

12.3.1 Development on Hunter 2

Compiled Microsoft BASIC programs can be developed on a **Hunter 2**, but the following considerations of memory size are important.

In addition to the source program text file, the following files are necessary:

Run Time Package	BRUN.COM	16K
Data For Linker	BCLOAD	2K
Linker	L80.COM	12K
Compiler	BASCOM.COM	32K
	Total:	62K

The compiler/linker operation generates two files, a relocatable and an object version of the program. Their size will each be roughly 50-80% of the program source. You can modify the source using the **Hunter 2** text editor Once the program is developed, space will also be needed for the related data files, in order to test the program.

12.3.2 Development on a Host Computer

CP/M 2.2 Computers

Since the **Hunter 2** operating system emulates CP/M 2.2, programs can be developed on a host computer running this operating system, and the resulting object file (.COM) loaded onto the **Hunter 2** for running and testing. BRUN.COM must also reside on the **Hunter 2**.

Other Host Computers

If the host computer supports a version of the Microsoft BASIC Compiler (eg MS-DOS or PC-DOS versions) then it is possible to create and compile programs which will be substantially compatible with the **Hunter 2** version of the compiler

Each compiler version will have certain features specific to the operating system or machine for which it is supplied. However, the bulk of the BASIC language is common. Generally functions relating direct to input and output, screen handling and graphics should be avoided when preparing a program for eventual use on a **Hunter 2**.

Having successfully compiled the program on the host machine, transfer the source file to a **Hunter 2** containing the Microsoft BASIC compiler system, and recompile and link. If the program has only used features common to both machines, this will be successful. If not, errors discovered when compiling on the **Hunter 2** must be corrected.

12.4 Turbo Pascal 3

Turbo Pascal 3 is distributed by Husky Computers as **Husky Turbo Pascal**. Versions are available for the CP/M 2.2 and MS-DOS operating systems. The notes in sections 3.1 and 3.2 (relating to avoiding version specific features) apply.

Turbo Pascal 3 programs can be developed on the **Hunter 2**, or on a host computer for which a version of Turbo Pascal is available. On a CP/M machine, including **Hunter 2**, only the file TURBO.COM is necessary at run-time, occupying 30K of disk space. For development, the additional files TURBO.OVR and TURBO.MSG should be present (about another 15K). Turbo Pascal 3 includes a text editor which is similar to WordStar, so an additional package is not required.

Compilation results in a .COM file, containing the object code and run-time library, whose size depends on the program size.

If development is undertaken on a host CP/M computer, care should be taken to avoid collision with the BDOS. The maximum end address available to programs must be set low enough to allow space for the loader, which resides just below BDOS when .COM files are run or executed from the TURBO system. The default setting is 700-1000 bytes lower than maximum memory (top of TPA). This should be changed to suit the lowest TPA size of the computers to be used.

When installing Turbo Pascal 3, you should supply the following terminal installation parameters:

Which Terminal	None
Send initialisation string	N
Send reset string	N
Cursor lead-in command	15
Cursor positioning between line and column	-
Cursor positioning between line and column	-
Column first?	Y
Offset to add to line	-
Offset to add to column	-
Binary address	Y
Clear screen command	1
Does clear screen also reset cursor?	Y
Delete line command	-
Erase to end of line command	-
Start of "Low Video"	-
Start of "Normal Video"	-
No. of rows on screen	24
No. of columns	40
Delay (3 times)	0
Is this definition correct?	Y
Operating frequency in MHz	4

12.5 C Compilers

'C' was developed with a very rigid definition of language and function, so that it would be easily usable on any computer with a UNIX operating system. In addition, various adaptations have been produced to allow development on popular microcomputer operating systems. A typical 'C' development system, MANX Software Systems' AZTEC, is available for the CP/M, MS-DOS, PC-DOS and other operating systems.

Like MBASIC, the full development system occupies a lot of space, and is designed for a disk based computer It comprises the 'C' Compiler, Assembler, Linkage Editor, development utilities and run-time libraries. Therefore, it is more suitable for use in development on a host computer, with the **Hunter 2** as the target, to which the executable .COM file (plus overlays if used) will be loaded for use. Full development requires space for a text editor, the program source text, the assembler file, the object file, the subroutine library and the executable file, in addition to the development system and data for testing. Even on a 496K **Hunter 2** this would severely limit the size of system that can be developed. An option is available to delete the assembler input file once used, which will save some space.

Despite the portability of the 'C' language, operation on different host and target machines requires care to ensure compatibility 'C' does not define any input/output operations, so these are provided as functions by the compiler vendor Development on a CP/M 80 machine for loading to a **Hunter 2** should not pose any problems. If development is undertaken on a 16 bit machine, for which a version of AZTEC 'C' is available, the proven source code must be transferred to the **Hunter 2** and compiled, assembled and linked there, space permitting. Alternatively, cross compilers are available from MANX to produce Z80 code from several host machines, including the IBM-PC.

Alternative compilers, such as the offering from Hi-Tech, provide code compatibility across operating systems and, in the case of Hi-Tech, an MS-DOS cross-compiler which produces code to run under CP/M 2.2.

12.6 Cross Compilers

The development environments described earlier rely on the availability of versions of the same language system for the host and target (**Hunter 2**) machines. This ensures that the language features and any libraries of functions are very similar, so that the number of inconsistencies is small and easy to avoid or cope with.

It is relatively easy to find microcomputer language systems with equivalent CP/M versions, because many were first developed for CP/M and later extended to 16 bit versions using operating systems which are based on CP/M (eg CP/M 86 or MS-DOS). However, this is not the case with most mini and mainframe computers.

Therefore, in order to develop CP/M software on a mini or mainframe host a cross compiler is used. This allows development using a language system for the host, the result of which is then converted by the cross compiler into executable instructions for the target processor (Z80 or equivalent).

There are several benefits to this method of development. The host computer is fast, and will employ powerful software for text processing and compilation, so that development time should be shorter than on a microcomputer However, a single user on a micro may achieve better turnround of work than one sharing the resources of a heavily loaded mini or mainframe.

Having developed the program, it can then be converted to work on several targets by processing through the appropriate cross compilers. So although cross compilers are usually expensive, in keeping with most software for large computers, they can automate the process of converting software from one environment to another, or producing several different versions of the software.

Typical examples of cross compilers suitable for the development of programs for the **Hunter 2** are:

Boston Systems Office	PASCAL on DEC VAX and PDP-11.
MANX AZTEC 'C'	'C' on UNIX, PC-DOS, MS-DOS.
Whitesmith Ltd	'C' or PASCAL on DEC VAX, PDP-11, LSI-11, IBM/370 MVS, VM/CMS, CP/M-86, PC-DOS.
Avocet Inc.	Z80, 64180 assembler and PASCAL on PC-DOS and MS-DOS.
Hi-Tech	'C' compilers on MS-DOS.
2500AD Software	Z80, 64180 assemblers on PC-DOS, MS-DOS. 'C' compilers on MS-DOS.
Pecon P System	PASCAL and MODULA 2 on PC-DOS, MS-DOS.
Hisoft	MODULA 2 on MS-DOS.

In selecting a cross compiler, it is very important to note that many are designed for development where the target is a board level device, rather than a general purpose microcomputer If the **Hunter 2** application will utilise files, then the cross compiler, like those above, must be compatible with the CP/M operating system, not just with Z80 microprocessor instructions.

12.7 Assemblers and Machine Code

There are several assemblers available for 8080 or Z80 instruction mnemonics, including ASM which comes free with CP/M 2.2 from Digital Research and Microsoft Macro Assembler. These can be run on the **Hunter 2**, or a desktop CP/M 2.2 machine, with no special installation or conversion problems.

Digital Research extended ASM to provide macro facilities, which enable the user to build up sets of instructions tailored to his own application. This product is MAC, which generates non-linkable hex files. A further update produced RMAC, which generates linkable relocatable output. RMAC also allows conditional assembly of sections of a source file. The DR linker is called LINK-80, and its output is a .COM file which can be run directly from DEMOS.

Microsoft's Macro Assembler is very well known and used, with powerful macro facilities, but it does not include a library of macros as does RMAC. Also it lacks a good debugging tool.

For debugging, Digital Research has produced products such as DDT (Dynamic Debugging Tool) or ZSID (Zilog Symbolic Instruction Debugger), and again these require no special installation. These are also useful for debugging code produced by cross compilers.

12.8 Software Manufacturers

Microsoft Corporation
10700 Northup Way
Bellevue,
Washington 98004
USA

Borland International Inc.
4585 Scotts Valley Drive
Scotts Valley,
California 95066
USA

MANX Software Systems
P O Box 55
Shrewsbury
New Jersey 07701
USA

Digital Research Inc.
P O Box 579
Pacific Grove
California 93950
USA

Boston Systems Office
128 Technology Centre
Waltham
Massachusetts 02254
USA

Whitesmiths Limited
97 Lawell Road
Concorde
Massachusetts 01742
USA

Avocet Systems Inc
120 Union Street
Rockport,
Maine 04856
USA

Hi Tech
PO Box 103
Alderley
Queensland 4051
AUSTRALIA

Hisoft
The Old School
Greenfield
Bedford
Bedfordshire
UK

12.9 Availability

Most of the systems and software products referred to above are available from your local software dealer If you have difficulty obtaining any product, please contact Husky Computers Ltd for details of your local supplier

Chapter 13 Assembly Language Programming

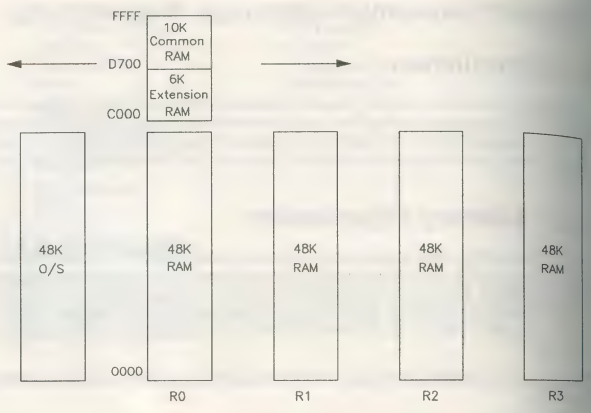
13.1 Introduction

This chapter describes how to program **Hunter 2** from assembly language, to produce COM programs or machine code routines to use from BASIC programs. It is not a detailed tutorial in assembly language programming or CP/M. If you require these, refer to any of the standard texts on the Z80 microprocessor and CP/M.

13.1.1 Memory Organisation

Hunter 2 has a large and complex memory map, consisting of RAM and ROMs. The RAM is used to store files, load and execute programs, store program data, operating system variables and buffers, the virtual screen contents, etc, etc. The ROMs (usually EPROMs) store the operating system, the **Hunter** BASIC Interpreter and optional permanent files such as additional communications protocols.

This memory is organised into pages of ROM and RAM, as shown in the following diagram:

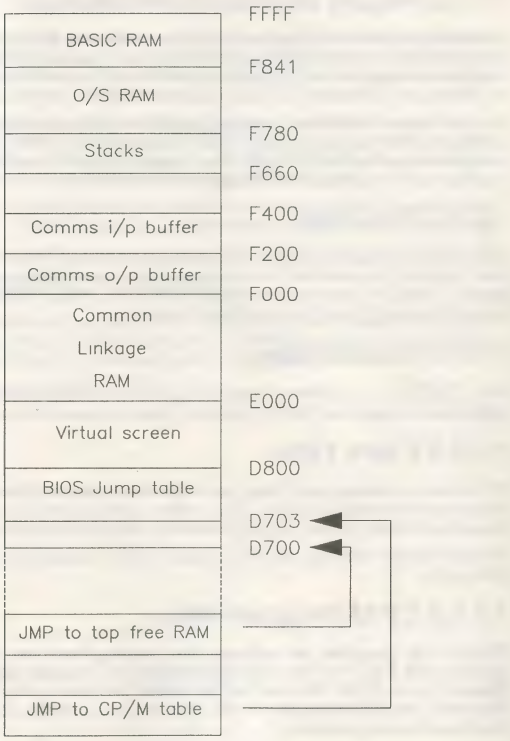


All pages except for R0, the execution memory (also known as the Transient Program Area, or TPA) are limited to 48k. R0 extends to 54k, although the top 6k together with the 10k of common RAM are never paged out. There is, therefore, up to 54k of RAM available for programming.

However, for many applications, you can ignore these complexities and consider the memory as a conventional CP/M memory map, like this:

The Common RAM is used to link together the various pages of RAM and ROM. It organises the paging and contains the jump tables for CP/M compatibility. The operation of paging is totally transparent to the user so **Hunter 2** is seen to have 54k program space with a powerful 48k operating system and BASIC Interpreter in ROM, and a block of filespace.

For a more detailed description of the useful locations in RAM page 0, see section 13.4.



13.1.2 Paging from Machine Code

In some applications the user may wish to access other pages from a machine code program. However, this is not a trivial matter and extreme care must be taken when using this facility Read and understand section 13.1.1 of this manual for full details of **Hunter 2** memory organisation before proceeding.

In order to use the **Hunter 2** paging, the machine code to switch the page **must** be located in the common block of RAM ie above 49152 (C000H). If the machine code is not in this block of RAM, the **Hunter 2** will not be able to access the machine code program once the page has been switched.

13.1.3 The Stack

The machine code stack pointer is initialised at switch on. It is not recommended that it should be moved as the system can use a considerable amount of stack space. Under no circumstances should routines be written which preclude the use of interrupts; serial I/O uses them liberally

13.1.4 The [PWR] Key

The [PWR] key is software controlled. The keyboard should be scanned periodically in any user written software if manual power off is required.

13.1.5 Cycle Time

Of interest in machine code loops, etc, is the execution time of the program. The cycle time in **Hunter 2** is 250ns. The number of cycles used by each instruction should be detailed in any good Z80 assembly language reference manual.

If serial data is received, then the routines will be considerably slower due to interrupts.

13.1.6 CP/M Programming

Object code programs are executed after loading them into the TPA. This provides 54k RAM for the program. Normally, the program will reside in the **Hunter 2** as a .COM file, and will be loaded and run by the File Manager. However, the File Manager is very versatile, and provides flexibility in file manipulation and execution.

13.1.7 File Organisation

Hunter 2's RAM disk is accessed exactly like a conventional disk, through the CP/M-compatible operating system calls. For most applications, this is all you should need to know about it. The following additional details are for information only.

The filespace pages are split into 2K byte sectors and allocated via a "sector allocation map" just like conventional disks. Files are accessed via a CP/M directory, divided into 16K extents. Any good text on CP/M file storage will give an adequate explanation, which is also true for **Hunter 2.**

This means that 2K byte blocks of a file are in no particular order within the filespace RAM, making large files, economical storage and fast updating possible.

13.1.8 Executing Programs

Programs are usually run by typing the file name in the File Manager If an HBA extension is typed, the file is loaded and run as a BASIC program, otherwise the extension will be forced to COM and a file of that name executed.

For more details, see Part 2.

13.2 Introduction to CP/M Programming

CP/M provides a universal interface, linking application programs into the operating system. Through a system of calls to one location (0005 Hex) with the CPU register C set to the call number, the ROM-based operating system in **Hunter 2** will implement the function and return to the user program. **Hunter 2** is therefore compatible with many readily available commercial products, conforming to these standard system calls.

13.2.1 System Calls Summary

CP/M system calls fall into two distinct categories: console (eg keyboard/LCD display) I/O and file I/O. Files are organised as CP/M disk files and are accessed via a file control block (FCB).

0 System Reset	19 Delete File
1 Console Input	20 Read Sequential
2 Console Output	21 Write Sequential
3 Reader Input	22 Make File
4 Punch Output	23 Rename File
5 List Output	24 Return Login Vector
6 Direct Console I/O	25 Return Current Disk
7 Get I/O Byte	26 Set DMA Address
8 Set I/O Byte	27 Get Allocation Address
9 Print String	28 Write Protect Disk
10 Read Console Buffer	29 Get R/O Vector
11 Get Console status	30 Set File Attributes
12 Return Version Number	31 Get Disk Params Address
13 Reset Disk System	32 Set/Get User Code
14 Select Disk	33 Read Random
15 Open File	34 Write Random
16 Close File	35 Compute File Size
17 Search for First	36 Set Random Record
18 Search for Next	37 Reset Drive

System calls 38-47 are not CP/M-compatible, except in principle. They are additional calls, provided to simplify access to **Hunter 2**'s additional features:

38 Set up RS232 (using system buffer)	46 Input Intel hex format
39 Set clock/calendar	47 Fetch key without echo
40 Write random with zero fill	48 Read clock/calendar
41 Set up RS232 (from menu)	49 Reserved
42 Get RS232 status	50 Force transmission
43 Select terminal emulation	51 Transmit handshake
44 Wand input	52 Reserved
45 Send Intel hex format	53 Reserved
	54 Serial output status

13.2.2 File Control Block

The FCB contains 36 bytes of information on the current file being accessed. On **Hunter 2**, the default FCB location is at 005CH, containing the following information:

Byte	Description
1	Drive code (zero)
2-9	Eight Character ASCII filename
10-12	ASCII file extension
13	Current extent number
14-15	Reserved
16	Record count for extent
17-32	Reserved for system use
33	Current record in sequential file operation
34-36	Random record number

The default DMA buffer is at 0080H.

13.3 Hunter 2 System Calls

This manual does not contain an exhaustive description of each system call, since the standard CP/M calls are defined and explained in great depth in many other publications. However, here is a brief description of each call, together with details of the extra calls specific to **Hunter 2**.

To use one of these calls, load the C register with the call number, load any other parameters required into the appropriate registers, and call address 0005H. Registers which are not mentioned are irrelevant on entry. **None of the registers are necessarily preserved on exit.**

To use these calls from BASIC, use CALL and ARG as described in Part 3.

13.3.1 System Reset (Call 0)

ENTRY	None.
EXIT·	None.
REMARKS:	A system restart, re-entering **Hunter 2**'s operating system.

13.3.2 Console Input (Call 1)

ENTRY	None.
EXIT·	A = ASCII character
REMARKS:	Read keyboard and echo onto screen. The call will wait until a key is pressed.

13.3.3 Console Output (Call 2)

ENTRY	E = ASCII character.
EXIT·	None.
REMARKS:	The character is displayed on the screen at the location of the cursor.

13.3.4 Reader Input (Call 3)

ENTRY	None.
EXIT·	A = ASCII character.
REMARKS:	The next character is read from the serial I/O buffer

13.3.5 Punch Output (Call 4)

ENTRY	E = ASCII character.
EXIT·	None.
REMARKS:	The character is sent out through the serial port.

13.3.6 List Output (Call 5)

ENTRY	E = ASCII character

EXIT· None.

REMARKS: As call 4.

13.3.7 Direct Console I/O (Call 6)

ENTRY E=0FFH to read a character, or E=ASCII character to output that character

EXIT· If used for input, A = ASCII character if there is one to input (ie in the console input buffer) or A=00H if there is no character If used for output, no results are returned.

REMARKS: Similar to calls 1 and 2, but there is no automatic echo to the screen.

This call will allow **Hunter 2** to enter POWER SAVE mode, if called repeatedly (eg from a loop). (POWER SAVE will be entered if the call is made more than 250 times in succession ie with no other BDOS calls in between.)

13.3.8 Get I/O Byte (Call 7)

ENTRY None.

EXIT· A = I/O status.

REMARKS: The present I/O status byte is returned.

13.3.9 Set I/O Byte (Call 8)

ENTRY E = I/O value.

EXIT· None.

REMARKS: I/O byte is set.

13.3.10 Print String to Console (Call 9)

ENTRY DE = string address.

EXIT· None.

REMARKS: An ASCII string whose start address is loaded into DE, is displayed on the screen. The string must be terminated with "$".

13.3.11 Read Console Buffer (Call 10)

ENTRY DE = buffer address.

EXIT· None.

REMARKS: The keyboard is read and the characters are placed into a buffer, the start address having been loaded into DE. The first byte contains the maximum number of characters to read into the buffer, the second byte contains the number of characters that have been placed into the buffer when input has been terminated with CR or LF

Console input is terminated when either the buffer overflows, or on Control J or Control M.

The read buffer is of the form:

DE	+0 mx	+1 nc	+2 c1	+3 c2	+4 c3	+5 c4	... ...	+N ?

where mx is maximum number of characters the buffer will hold (1 to 255).

nc is the number of characters read (set by DEMOS on return). nc is followed by c1, c2, etc., the characters read from the console.

If nc < mx then uninitialised positions follow the last character (represented above by ?).

The character after the last read character is set to 00H, for compatibility with this undocumented CP/M feature. (This is not true for versions before 9G08.)

The following control functions are recognised during line input:

Control C	reboots system if at start of line
Control E	causes physical end of line
Control H	backspace one character position
Control J	terminate input
Control M	terminate input
Control R	retype current line after new line
Control X	backspaces to beginning of current line

13.3.12 Get Console Status (Call 11)

ENTRY. None.

EXIT: A = console status.

REMARKS: If a character has been typed then 01H is returned, 00H if not.

NOTE: This call does not allow **Hunter 2** to enter POWER SAVE mode, so may decrease battery life if used repeatedly (eg in a loop). Use Call 1, 10 or 47 instead, to allow **Hunter 2** to enter POWER SAVE mode.

13.3.13 Return Version Number (Call 12)

ENTRY. None.

EXIT: HL = version number.

REMARKS: **Hunter 2** is CP/M version 2.2 compatible, so 0022H is returned.

13.3.14 Reset Disk System (Call 13)

ENTRY· None.

EXIT· None.

REMARKS: The filing system is reset, and the DMA address is returned to 80H.

13.3.15 Select Disk (Call 14)

ENTRY None.

EXIT· None.

REMARKS: **Hunter 2** has "RAM disks" so this call produces no action.

13.3.16 Open File (Call 15)

ENTRY· DE = FCB address.

EXIT· A = directory code.

REMARKS: The file specified by the FCB is opened. If successful A = 0, 1, 2 or 3. If not successful, A = 0FFH.

13.3.17 Close File (Call 16)

ENTRY DE = FCB address.

EXIT· A = directory code.

REMARKS: If successfully closed, A = 0, 1, 2 or 3. If not successful, A = 0FFH.

13.3.18 Search for First (Call 17)

ENTRY DE = FCB address.

EXIT· A = directory code.

REMARKS: The directory is searched for the first match of the FCB. A = 0FFH if there is no match, and 0,1,2 or 3 if there is. The filename is returned into the current DMA buffer The search function returns the complete entry from the buffer

13.3.19 Search for Next (Call 18)

ENTRY None.

EXIT· A = directory code.

REMARKS: Similar to CALL 17, except that the next entry is looked for The search function returns the complete entry from the buffer

13.3.20 Delete File (Call 19)

ENTRY DE = FCB address.
EXIT· A = directory code.
REMARKS: Files which match the FCB are deleted. If none exist A = 0FFH is returned.

13.3.21 Read Sequential (Call 20)

ENTRY· DE = FCB address.
EXIT· A = directory code.
REMARKS: The specified file is read, and the sector copied to the current DMA address. A = 00 for a successful read.

13.3.22 Write Sequential (Call 21)

ENTRY DE = FCB address.
EXIT· A = directory code.
REMARKS: The record at the current DMA address is written to the disk. A = 00 to indicate a successful write.

13.3.23 Make File (Call 22)

ENTRY· DE = FCB address.
EXIT· A = directory code.
REMARKS: The specified file is created. A = 0-3 if successful, or A = 0FFH if not.

13.3.24 Rename File (Call 23)

ENTRY DE = FCB address.
EXIT· A = directory code.
REMARKS: The file specified in the first 16 bytes of the FCB is renamed to that name in the second 16 bytes.

13.3.25 Return Login Vector (Call 24)

ENTRY None.
EXIT· HL = login vector.
REMARKS: **Hunter 2** returns HL = 1 and A = 1, indicating a single disk system.

13.3.26 Return Current Disk (Call 25)

ENTRY None.
EXIT· A = current disk.
REMARKS: Zero is always returned on **Hunter 2**.

13.3.27 Set DMA Address (Call 26)

ENTRY DE = DMA address.
EXIT· None.
REMARKS: The current disk data buffer area is set to DE.

13.3.28 Get Allocation Address (Call 27)

ENTRY None.
EXIT· HL = allocation address.
REMARKS: Zero only is returned.

13.3.29 Write Protect Disk (Call 28)

ENTRY
EXIT·
REMARKS: Returns with no effect.

13.3.30 Get Read Only Vector (Call 29)

ENTRY
EXIT·

REMARKS: Returns with no effect.

13.3.31 Set File Attributes (Call 30)

ENTRY DE = FCB address.
EXIT· A = directory code.
REMARKS: File attributes for read only etc. are set or reset as specified.

13.3.32 Get Disk Parameter Address (Call 31)

ENTRY None.
EXIT: HL = disk parameter block address.
REMARKS: Disk parameters are returned which indicate a double sided double density disk. This is to allow for expansion within **Hunter 2**. It should not be used for space calculation purposes.

13.3.33 Get or Set User Code (Call 32)

ENTRY·
EXIT:
REMARKS: Returns with no effect.

13.3.34 Read Random (Call 33)

ENTRY DE = FCB address
EXIT· A = directory code.
REMARKS: The file is read at the specified record. Normally non zero directory codes mean an error

13.3.35 Write Random (Call 34)

ENTRY DE = FCB address.
EXIT· A = directory code.

REMARKS: The file is written at the specified record. A = 0 means a successful write.

13.3.36 Compute File Size (Call 35)

ENTRY DE = FCB address.
EXIT· Random record field set to one plus the last record.
REMARKS: Each record is 128 bytes, so the file size is 128 * (record field value - 1).

13.3.37 Set Random Record (Call 36)

ENTRY DE = FCB address.
EXIT· Random record field set.
REMARKS: Normally used to set the random record position after reading or writing sequentially, to allow random access.

13.3.38 Reset Drive (Call 37)

ENTRY DE = drive vector.
EXIT· None.
REMARKS: **Hunter 2**'s filing system is reset.

13.3.39 Set up Serial Port (Call 38)

ENTRY None.
EXIT· None.
REMARKS: The serial port is set up using the contents of the serial port setup buffer (TXSPEED to RXPROT; see section 13.4 for details).

13.3.40 Initialise Clock (Call 39)

ENTRY None.
EXIT· None.

REMARKS: The time and date buffer RAM (TENTHSEC to TENYR; see section 13.4 for details) is used to initialise the clock.

13.3.41 Write Random with Zero Fill (Call 40)

ENTRY. DE = FCB address

EXIT: A = directory code

REMARKS: The write random with zero fill operation is similar to function 34, with the exception that a previously unallocated block is filled with zeros before data is written.

13.3.42 Initialise Communications (Call 41)

ENTRY· None.

EXIT· None.

REMARKS: The serial port setup menu is entered (as described for the COMS command, in Part 2). When the user leaves the menu, the call returns to the calling routine, not to the File Manager

13.3.43 Get Serial Status (Call 42)

ENTRY· None.

EXIT: A = port status.

REMARKS: To test for pending serial input characters. On return, A=0 for no characters read else A is non-zero

13.3.44 Terminal Emulation Call (Call 43)

ENTRY None.

EXIT· None.

REMARKS: Enters terminal emulation mode

13.3.45 Wand Input Call (Call 44)

ENTRY DE = Buffer Address.

EXIT· None.

REMARKS: The Wand code is selected previously by setting the BARTYPE location according to wand code being read.

The code is read into the buffer. The user must provide the buffer address and, in the first location of the buffer, the user must provide the maximum number of characters to be read into the buffer. On exit, the second byte of the buffer contains the number of characters that have been placed in the buffer, followed by the wand character's input.

13.3.46 Hex Format Output (Call 45)

ENTRY IX = start of data.
DE = end of data.

EXIT· None

REMARKS: The data is converted to Intel Hex format (see Appendix VII) and sent out through the serial port.

13.3.47 Hex Format Input (Call 46)

ENTRY IX = *limit*.

EXIT· Intel Hex format data blocks (see Appendix VII) are input from the serial port, converted back to binary and stored at the locations specified by the blocks, as long as the addresses are less than *limit*.

13.3.48 Fetch Key without Echo (Call 47)

ENTRY None.

EXIT· A = ASCII code of character.

REMARKS: If there is no key to read in the input buffer, this call waits for keyboard input, in POWER SAVE mode. When the input is received, it is not echoed to the display

13.3.49 Read Clock (Call 48)

ENTRY None.

EXIT: None.

REMARKS: The time and date is copied from the clock into the time and date buffer RAM (see section 13.4), locations TENTHSEC to TENYR.

13.3.50 Reserved (Call 49)

This call is not available to the user.

13.3.51 Force transmission (Call 50)

ENTRY None.

EXIT· None.

REMARKS: This function forces the transmission of the last block in protocols such as SYSTIME and IBM3780, which have a timeout facility on the communications output buffer

13.3.52 Transmit handshake (Call 51)

ENTRY: E = handshake character

EXIT· None.

REMARKS: This function outputs the handshake character contained in E via the serial port.

13.3.53 Reserved (Call 52)

This call is not available to the user

13.3.54 Reserved (Call 53)

This call is not available to the user

13.3.55 Serial output status (Call 54)

ENTRY None.

EXIT· A = serial port status.

REMARKS: If there are no characters in the serial output buffer, then A = 0, otherwise A is non-zero.

13.4 Memory Locations

Location (decimal)	Location (hex)	Name	Function
55043 55046 55049 55052 55055 55058 55061	D703 D706 D709 D70C D70F D712 D715	JMP WARMSTART JMP CONSTAT JMP CONIN JMP CONOUT JMP LIST JMP PUNCH JMP READER	CP/M BIOS jump table
55296 -55424	D800 -DF7F	VERSCRN	Virtual screen 80 x 24 characters. D800 is the top left-hand corner
57280	DFC0	ALPHALOK	In caps lock mode, SHIFT plus a letter key will normally produce a lowercase letter. If ALPHALOK is set to non-zero, an uppercase letter will be typed instead.
57344 -61439	E000 -EFFF		Operating system area.
61440 -62463	F000 -F3FF		Communications input and output buffers.
62981 -63031	F605 -F637	USER	This area is left free by the operating system and is available for use by user machine code routines.

Location (decimal)	Location (hex)	Name	Function
63168 -63359	F6C0 -F77F		Operating system stacks.
63360	F780	IPFLAG	Logical keyboard flag: 0 = **Hunter 2** keyboard or LCD screen 128 (80H) = Serial I/O port
63364	F784	VECTOR	HELP vector address. The key is trapped by the keyboard software and is vectored to this location.
63394	F7A2	QUIET	Flag to control keyclick. If non-zero, keyclick is inhibited.
63415	F7B7	APHRO	Flag to indicate how **Hunter 2** was powered up: 0 = Keyboard 1 = Serial port 2 = Clock
63416	F7B8	NYMPHO	If the standard power down routine is used then setting this byte to non-zero will inhibit operation of the power off key ([PWR]). **NOTE:** This byte is cleared on power up.
63419	F7BB	FOREVER	Setting non-zero prevents automatic switch off after time-out.

Location (decimal)	Location (hex)	Name	Function
63464	F7E8	TENTHSEC	Time memory 1/10 sec.
63465	F7E9	UNITSEC	" " unit sec.
63466	F7EA	TENSEC	" " tens sec.
63467	F7EB	UNITMIN	" " unit mins.
63468	F7EC	TENMIN	" " tens mins.
63469	F7ED	UNITHRS	" " unit hours.
63470	F7EE	TENHRS	" " tens hours.
63471	F7EF	UNITDAY	" " unit days.
63472	F7F0	TENDAY	" " tens days.
63473	F7F1	UNITMTH	" " unit month.
63474	F7F2	TENMTH	" " tens month.
63475	F7F3	UNITYR	" " unit years.
63476	F7F4	TENYR	" " tens years.
63480	F7F8	TXSPEED	Set serial O/P baud rate: 1 = 75 4 = 300 7 = 1800 2 = 110 5 = 600 8 = 2400 3 = 150 6 = 1200 9 = 4800
63481	F7F9	RXSPEED	Set serial I/P baud rate (same as TXSPEED).
63482	F7FA	CTSAF	The CTS enable flag: 0 = No 1 = Yes
63483	F7FB	DTRAF	The DTR enable flag: 0 = No 1 = Yes
63484	F7FC	RTSAF	The RTS enable flag: 0 = No 1 = Hold 2 = True
63485	F7FD	DSRAF	The DSR enable flag: 0 = No 1 = Yes
63486	F7FE	DCDAF	The DCD enable flag: 0 = No 1 = Yes

Location (decimal)	Location (hex)	Name	Function
63487	F7FF	TXPTY	Transmit parity flag: 0 = None 1 = Odd 2 = Even 3 = 8-bit, no parity 4 = 8-bit, odd parity 5 = 8-bit, even parity (4 and 5 on 9G08 and later only)
63488	F800	RXPTY	Receive parity flag: 0 = None 1 = Odd 2 = Even 3 = 8-bit, no parity 4 = 8-bit, odd parity 5 = 8-bit, even parity (4 and 5 on 9G08 and later only)
63489	F801	TXPROT	Transmission protocol: 0 = None 1 = XON/XOFF 2 = ETX/ACK 3 = ACK/NAK 4 = SYSTIME
63490	F802	LFACT	Line feed active: 0 = No 1 = Yes
63491	F803	NULAF	Number of NULLs following CR/LF: 0 = None 1 = Two 2 = Five 3 = Ten 4 = Twenty
63492	F804	ECHOF	The serial output echo flag: 0 = No echo (Full Duplex) 1 = Output echoed to **Hunter 2**'s screen

Location (decimal)	Location (hex)	Name	Function
63493	F805	RXPROT	Receive protocol (as TXPROT).
63495 -63496	F807 -F808	CTSVECT	CTS interrupt vector These locations contain an address for the CTS input.
63502	F80E	ESCODE	Escape code. Used when escaping from fully running code. Used to interrupt execution of user programs when BASIC auto-start is in use. ASCII characters are stored in Decimal form, most significant digit at lowest memory location.
63507	F813	STARTFL	Flag to indicate immediate running of user BASIC program: AAH = Immediate run.
63551 -63552	F83F -F840	DEFLIM	Address of the lower limit of BASIC Symbol table (End of free Memory).
63553 -63554	F841 -F842	ALIM	Beginning of free memory after BASIC source and arrays.
63563	F84B	SERIG	This contains the ASCII code of a character to ignore when receiving serial data. If not required set to 80H (128 decimal).
63567	F84F	VEIWANG	D0-D3 give the current viewing angle setting, divided by 4. D7 gives the backlight setting (0 = off).

Location (decimal)	Location (hex)	Name	Function
63571	F853	RECLEN	(IBM 2780.) This is a single byte defining maximum record length.
63572	F854	BUFLEN	(IBM 2780.) This is a two byte number defining the maximum block length.
63574	F856	SRFLG	(IBM 2780.) This flag controls single or multiple records in a block: 0 = multiple records non-0 = single record per block
63575	F857	RPFLG	(IBM 2780.) A single byte flag used to control record padding: 0 = no padding non-0 = padding
63576	F858	EMFLG	(IBM 2780.) Determines record terminator used, the essential differences between 2780 and 3780 communications: 0 = 2780 mode non-0 = 3780 mode
63577	F859	STXFLG	(IBM 2780.) A single byte flag controlling use of intermediate STX: 0 = STX at start of first record non-0 = STX at start of each record
63578	F85A	RETRYS	(IBM 2780.) A single byte flag controlling error handling by the protocol: 0 = up to 10 retries before error non-0 = retry without error

Location (decimal)	Location (hex)	Name	Function
63579	F85B	SCFLG	(IBM 2780.) A single byte flag controlling operation of space compression: 0 = no compression/expansion non-0 = compression/expansion
63584	F860	BARTYPE	Select Barcode type: 0 = CODE 39 1 = EAN8/13 2 = Int 2 of 5
63585	F861	TOPAGE	Maximum number of pages in **Hunter 2**. **NOTE:** Bit 7 is set to indicate RAM.
63889	F991	LSTBSN	CAPS lock: 0 = OFF 1 = ON **NOTE:** Must be 0 or 1
65266	FEF2	KEYBUF	112 bytes of keyboard RAM, see section 13.6.
65378	FF62	SPELFLG	Keyboard changed flag: 0 = keyboard layout changed
63420	F7BC	COMERR	Location which maintains the error code for a communications error This location is 0 if there has been no error.
63581	F85D	RXTOAF	Receive timeout active (parameters as TXTOAF).
63582	F85E	TXTOAF	Transmit timeout: 0 = no timeout 1 = 10 sec timeout 2 = 20 sec timeout 3 = 30 sec timeout 4 = 40 sec timeout 5 = 50 sec timeout 6 = 60 sec timeout

13.5 I/O Address Map

Hunter 2 uses the NSC800 port map for all its I/O functions, including the keyboard, RS232 port, etc. The following gives a list of the most useful ports which can be controlled from machine code (or BASIC, using OUT and INP).

Address (Decimal)	Address (Hex)	Name	Description
1	1H	DSR	Bit seven is the DSR input bit: 0 = Active 1 = Inactive
2	2H	INPUTS	This port has a number of input functions: Bit 0 = Data in (inverted from RS232 line) Bit 1 = DCD in (inverted from RS232 line) Bit 2 = Power low warning (0 = ok, 1 = low) Bit 3 = TXCLK in, also the single input bit.
79	4FH		Programs the RTC to generate an interrupt after a delay 0 = disable interrupt 1 = 0.5 second delay 2 = 5 second delay 4 = 60 second delay
96	60H	ANGLE	Bit 1 controls the optional backlight: 0 = off, 1 = on Bits 2-5 control the display viewing angle. The higher the value, the darker the display.
129	81H	V24OUT	Directly outputs to the V24 data line signal on bit 0. The output is voltage inverted ie 0 = +ve 1 = -ve

Address (Decimal)	Address (Hex)	Name	Description
130	82H	RTS	RTS output bit
131	83H	POWHLD	Power control: bit 0 = 1 **Hunter 2** is on bit 0 = 0 **Hunter 2** is off
132	84H	INVCON	V24 invertor control: bit 0 = 1. invertor on bit 0 = 0: invertor off
133	85H	DTR	DTR output bit
135	87H	AUXPWR	Turns on the auxiliary power to pin 10 on the 'D' type connector· 1 = Power on 0 = Power off
187	BBH	ICRREG	NSC800 internal interrupt mask register See the NSC800 handbook
224	E0H	BPAGE	Memory paging register

13.6 Redefining the Keyboard

The effect of pressing each keyboard key is defined by 'key codes' stored in 114 consecutive RAM locations named KEYBUF These memory locations form a 'map' of the physical keyboard. The upper shift keys are defined in the first 56 locations, followed by the 56 lower shift keys, as shown in the following diagram.

On power up, the operating system resets the keyboard map to produce the standard keyboard configuration, but a user program can change the keyboard map to produce special keyboard arrangements.

The key codes fall into two main groups: ASCII codes and Special codes.

ASCII codes are in the range 0 to 7FH, which cause the keys to type Control Codes (such as CR), letters, numbers and other characters. See Appendix III for a description of ASCII codes.

Special codes are in the range 80H to FFH, which have a number of special effects, described in 13.6.1 below

!	@	#	$	%	^	&	*	(	)	—	+	↑	ESC	PWR
65278	65271	65279	65272	65273	65266	65274	65267	65275	65268	65277	65270	65276	65269	65379
65334	65327	65335	65328	65329	65322	65330	65323	65331	65324	65333	65326	65332	65325	65380
1	2	3	4	5	6	7	8	9	Ø	–	=	↑	BRK	HLP
Q	W	E	R	T	Y	U	I	O	P	]	←	LBL	→	
65292	65285	65293	65286	65287	65280	65288	65281	65289	65282	65291	65284	65290	65283	65297
65348	65341	65349	65342	65343	65336	65344	65337	65345	65338	65347	65340	65346	65339	
q	w	e	r	t	y	u	i	o	p	[	←	INS	→	ENTER
CTL	A	S	D	F	G	H	J	K	L	:	"	↓	DEL	
65306	65299	65307	65300	65301	65294	65302	65295	65303	65296	65305	65298	65304	65311	
65362	65355	65363	65356	65357	65350	65358	65351	65359	65352	65361	65354	65360	65367	65353
FN	a	s	d	f	g	h	j	k	l	;	,	↓	BS	
⇧	Z	X	C	V	B	N	M	SPACE		<	>	?	\	⇧
65320	65313	65321	65314	65315	65308	65316	65309	65317		65310	65319	65312	65318	65320
65376	65369	65377	65370	65371	65364	65372	65365	65373		65366	65375	65368	65374	65376
⇧	z	x	c	v	b	n	m	SPACE		,	.	/	TAB	⇧

13.6.1 Special Codes

There are special codes to generate the following effects:

(1) CTL/FN
(2) ⇧ (latched)
(3) HELP
(4) Momentary ⇧
(5) PWR
(6) Virtual screen window up key
(7) Virtual screen window down key
(8) Virtual screen window right key
(9) Virtual screen window left key
(10) LBL
(11) INS
(12) Increment viewing angle
(13) Decrement viewing angle
(14) View right-hand side of virtual screen
(15) View left-hand side of virtual screen
(16) Caps lock
(17) Backlight toggle
(18) Momentary backlight

If you change the Control or Shift codes, to make the operating system use the changed keys, clear the SPELFLG location (ie set it to 00H).

ENTER is not a special code: it is simply CR (0DH).

Name	Code	Effect
Latched Shift	129 (81H)	This key should be defined in both shift halves in order to function correctly! This function gives a 'toggle' action, changing shift with each operation.
Momentary Shift	130 (82H)	As **Hunter 2** powers up in the lower shift, it is important for the momentary shift code to be defined in the lower shift half of KEYBUF This function gives a 'conventional' shift requiring simultaneous depression of the ⇧ and the key to be shifted.
Control	132 (84H)	If this key is held down, the code for another key is derived from the code located in the lower shift half of KEYBUF, if this key has a valid 'control' equivalent.
Control	133 (85H)	If this key is held down, the code for another key is derived from the code located in the upper shift half of KEYBUF, if this key has a valid 'control' equivalent.
HELP	134 (86H)	Enter HELP text mode. Help mode is exited by pressing [HELP] again.
ON/OFF	136 (88H)	Act as [PWR]. After **Hunter 2** has been powered off the power key defaults back to [PWR] ie the top right hand position of the keyboard.

Name	Code	Effect
Virtual Screen Window Up Virtual Screen Window Down Virtual Screen Window Right Virtual Screen Window Left	138 (8AH) 139 (8BH) 140 (8CH) 141 (8DH)	Move the display window over the contents of the virtual screen. These keys are completely transparent to the user program.
Label	143 (8FH)	Switch the function key display line on and off. The key is transparent to the user program.
Insert	144 (90H)	Return the virtual screen window to its original position after being moved by the window control keys. The key is transparent to the user program.
Increment viewing angle Decrement viewing angle	146 (92H) 147 (93H)	Control the LCD contrast, to adjust for different viewing angles.
View rhs of virtual screen View lhs of virtual screen	148 (94H) 149 (95H)	Flip the virtual screen window to show the leftmost or rightmost 40 characters.
Caps lock	150 (96H)	Toggle the alphabetic keys to type capitals when operated without ⇧.
Backlight toggle*	151 (97H)	Toggle optional backlight on and off.
Backlight*	152 (98H)	Turn optional backlight on while key depressed.

*These codes are not available in versions before 9G08.

13.7 Controlling Power-Off

In some machine code programs it may be necessary to power the **Hunter 2** off under program control. This can be achieved by an OUT instruction to the port POWHLD (83H). However, if the program is to be executed in CONT mode, it must ensure that a correct power down procedure is followed and cannot use this simple method.

In CONT mode, the power off sequence is controlled by the operating system to ensure that the contents of registers prior to the power off are preserved and that on restoring power the program can continue operation. The user may modify the power key detection process by modifying the OFFVECT locations, to point to the user's power off routine.

The OFFVECT location is two RAM locations containing a 16 bit pointer, lower byte stored first, which normally points to the power routine in the operating system which is executed when a power key is detected. The user may re-vector this to his own 'power key' handling routine with care.

If the user wishes to power off the **Hunter 2** under software control, a CALL must be made to the following code which should be located in a common area of RAM. Note that the value of POWER is dependent on the operating system version.

```
BPAGE       EQU     0E0H        ; PAGE LATCH
POWER       EQU     0E4C7H      ; POWER OFF ROUTINE (9608)
;POWER      EQU     0E4D1H      ; POWER OFF ROUTINE (9606)
PAGE        EQU     0F7C6H      ; PAGE STORAGE REGISTER
OFFVECT     EQU     0F7B9H      ; POWER KEY DETECTION VECTOR
NYMPHO      EQU     0F7B8H      ; INHIBIT POWER OFF FLAG
;
TURNOFF:    XOR     A           ; Clear A
            LD      (NYMPHO),A
            LD      (PAGE),A
            OUT     (BPAGE),A   ; select EPROM
            CALL    POWER
TURNOFF1:   LD      A,80H       ; select TPA
            LD      (PAGE),A
            OUT     (BPAGE),A
            RET
```

For example:

```
.
;previous code
.
            CALL    TURNOFF     ; Power off machine
.
;code after powering on
.
```


The TURNOFF routine selects the EPROM page and then executes the power off routine in the operating system. On power up, if in CONT mode, the program continues execution at TURNOFF1 and selects the transient program area page to return back to the main program.

If the user wishes to execute the application program from the beginning after power up, the following option may be used:

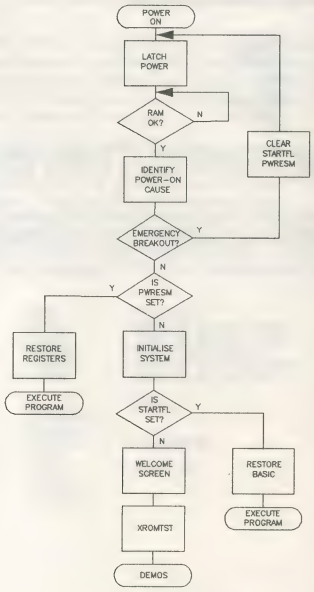
```
TURNOFF:    XOR     A
            LD      (PAGE),A
            OUT     (BPAGE),A
            LD      (NYMPHO),A
            CALL    POWER
TURNOFF1:   LD      A,80H
            LD      (PAGE),A
            OUT     (BPAGE),A
            JP      100H
```

In the above example, after selecting the TPA page, program execution jumps to commence at 100H in the TPA which is the entry point for normal CP/M programs.

This information only applies to program-controlled power off. Manual timeout or battery interruption power off sequences are transparent to the user.

13.8 Power-up

Hunter 2 is normally turned on by pressing the [PWR] key The actions taken on power-up are summarised in the following flow diagram:



13.9 Using EPROMs for Programs (9G08 only)

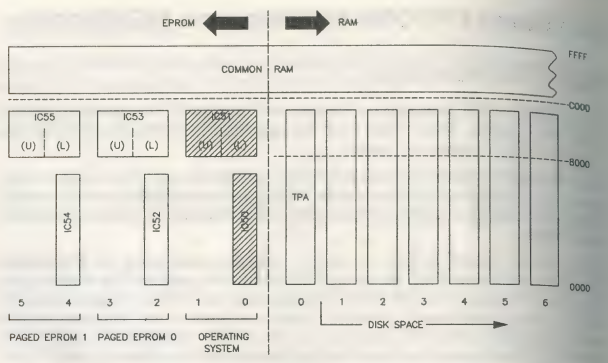
13.9.1 Introduction

This facility is NOT available on **Hunters** with version 9G06 operating system, or earlier versions. These units must be upgraded to 9G08 first, by returning them to Husky Computers Limited.

While **Hunter 2**'s battery-backed RAM disk provides fast and convenient storage for most applications, its reliance on the battery and susceptibility to being erased, either accidentally or deliberately, may be a disadvantage in certain specialized applications.

This section describes how to store programs and data in Electrically Programmable Read Only Memory (EPROM). Data may only be written to an EPROM by means of an EPROM programming unit. Once programmed, the data is retained indefinitely, whether power is applied or not. Data may only be erased from the device by exposure to ultra-violet radiation of a particular wavelength band over a period of time. Consequently the device is used for the storage of permanent programs, such as the operating system.

The following diagram shows the memory map of a **Hunter 2** fitted with 27C256K EPROM devices. The diagram shows that in addition to the paged RAM, EPROM pages are available. This memory is capable of providing the user with two 64K EPROM memory areas for the retention of program or data.



Pages 0 and 1 of the EPROM area are reserved for use by the operating system.

13.9.2 EPROM Application

Whilst main and stand-by batteries will protect memory content for very long periods, (typically several years for alkaline batteries, several months for NiCad batteries), prolonged storage could theoretically lead to loss of retention voltage and subsequent data corruption. The Husky EPROM option is designed to allow long term (typically ten years) retention of user application programs regardless of memory support power supply status.

There is also a very small possibility of data corruption occurring in normal use of the computer through so called 'soft' errors that can occur because of the effect of the decay of certain radio-isotopes that are naturally present in semi-conductor materials and integrated circuit assemblies.

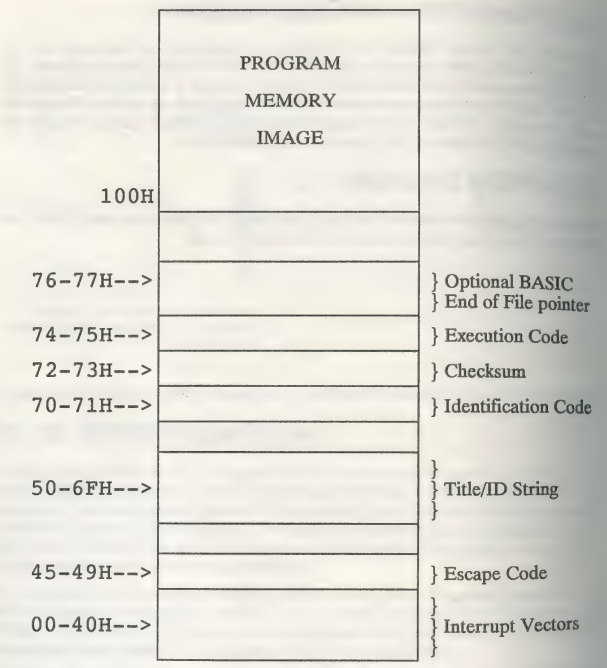
Whilst loss of data for these reasons is extremely rare, Husky recognise that some users may wish to 'back-up' their application programs in EPROM devices so that the computer is guaranteed to be available for use on power-up, regardless of storage or other circumstances.

13.9.3 Description of EPROM Option

The EPROM option allows the copying, on command or automatically on power-up, of the contents of the user EPROM (or EPROMs) into the **Hunter 2**'s TPA. The contents of the user EPROM may be either a BASIC tokenised file, a CP/M application program or user data such as look-up tables. Due to the architecture of the **Hunter 2**, the largest single program which may be transferred from the EPROM area is 53K bytes of contiguous code.

13.9.4 ROM Structure

The following diagram shows the configuration of the lower section of the ROM memory The seven main sections are described below



Interrupt Vector 0-40H

This area contains the interrupt vector area as found in all memory pages. It is essential that this section of memory is present as interrupts can occur at any time and, therefore, in any page of the system.

Escape Code 45H-49H

When the EPROM image is loaded into the TPA, either on command or automatically, the system is placed into 'CONT' mode. In order to escape from this mode the security escape code must be used as described in the manual (File Manager 'CONT' command, described in Part 2). The EPROM option automatically copies the security key sequence into the operating system from this area. This area must contain ASCII values which are available from the keyboard during program operation in order to operate correctly.

Also the function 'Control C' must be available from the keyboard during program execution, otherwise break-out from the program will not be possible preventing essential testing of the operating system and machine.

Program Title 50H-6FH

Up to 32 bytes are available to store an ASCII program title and or program version number. This section is a free format ASCII field which can be defined at the users discretion and is printed in response to the ROM VER n DEMOS command. It is intended to provide a simple means for the user to identify the program and version fitted in user EPROM.

Identification Code 70-71H

This double byte area is used exclusively by the operating system in order to determine the presence of valid EPROM option EPROMs fitted to the machine, as follows:

70H	71H	Interpretation
0BBH	00H	COM file
0BBH	0F1H	BASIC file

Checksum 72H-73H

A checksum is generated and stored on the 64K EPROM memory block in order to verify, on loading, the integrity of the data once loaded into the TPA prior to execution. The checksum is a 16 bit unsigned integer which is the sum of the bytes in the bootable program from 100H to its end.

Execution Code 74H-75H

This two byte code signifies one of the following options to the operating system:

74H	75H	Interpretation
00H	00H	Loading and execution of the EPROM software will occur on the command 'ROM n' from DEMOS but formatting of the machine will not occur
00H	01H	Formatting of the machine followed by loading and execution of the EPROM software will occur on the command 'ROM n' from DEMOS.
01H	00H	Loading and execution of the EPROM software will occur on power up but no formatting of the machine will occur.
01H	01H	Automatic formatting of the machine followed by loading and execution of the EPROM software will occur on power up.

BASIC Pointers 76H-77H

These bytes are used exclusively by the operating system when it loads a BASIC tokenised program or COM program from the EPROM area into the TPA. They contain a pointer to the end of the BASIC source or the COM file. When loading a BASIC program, the interpreter uses them to find the end of the program.

13.9.5 Description of Operation

The EPROM image is copied either automatically on power-up, or by command from DEMOS. The load automatically on power-up option is determined by the special code which is present in the execution bytes of the EPROM image itself.

This option is available only in the ROM 0 area of paged EPROM memory, however, the ROM n DEMOS command allows execution of either of the two EPROM images if equipped.

The special code present in the execution bytes of the EPROM image also defines if the machine is to be formatted before the EPROM image is loaded and executed (either automatically on power-up or on the command 'ROM n' from DEMOS). A format will ensure that the computer's status is 'clear', initialising buffers, pointers and other settings to nominal values, before execution of the EPROM image. This implies that any RAM contents previously present will be lost, but also ensures that any random corruptions which may have occurred during storage or other circumstances cannot disrupt operation of the machine.

The special codes etc. are set during the generation of an EPROM image according to the users selections.

The user's application program must ensure that any parameters utilised by the application, such as the real-time clock, or specific communications parameters, are re-initialised by the application program on power-up, or after execution of the ROM n command.

The ROM contents are always copied to address 100H upwards of the TPA for compatibility with the CP/M memory structure. Once loaded into the TPA, a checksum is performed on the TPA area which has been loaded, and this is then verified against the checksum bytes of the EPROM image. If a failure is detected, the message 'ROM failure - misload' is displayed on the screen and the user is returned to DEMOS after the TPA image has been 'wiped clean', at this stage the machine can be powered off and will not yet be in 'CONT' mode.

If ROM load occurs successfully, the computer will be placed in 'CONT' mode and program execution will commence. The escape code in the EPROM image will automatically be copied into the operating systems escape sequence buffer prior to execution of the program.

This escape sequence allows essential testing and debugging of the operating system to be performed prior to despatch of the computer from the factory, and also to allow any maintenance action at any other time without compromising the security of the user's program or relying on the users program itself for test purposes.

The user EPROM images are best installed by Husky Computers, where extensive quality and performance checks are carried out. It is essential that the installation is carried out by trained personnel because the **Hunter 2**'s vital weatherproofing depends on careful assembly

13.9.6 Extra DEMOS Commands

ROM n

This command copies the EPROM image of page *n* into the TPA and executes it. The contents of the program are sum checked and, if successful, the program is executed in 'CONT' mode. *n* is 0 or 1.

Note that as with the automatic option, the contents of the computer may be formatted prior to the execution of the command to ensure that the computer's status is 'clear' before execution of the EPROM image.

ROM VER n

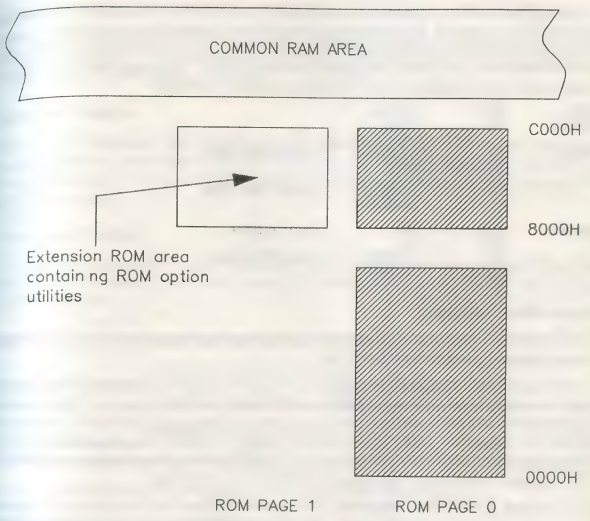
This command returns the program title field of the EPROM image selected by *n* (Note the security code is not returned via this command unless it is included specifically by the user in the title field.) *n* is 0 or 1.

GENERATE

This command allows the user to enter the EPROM generation routine. This routine, described later, prompts the user to enter the information needed to allow the program to generate the Intel Hex version of the EPROM image of the program contained in the TPA.

13.9.7 Machine Code Linkages

The operating system organisation is as follows:



Page 1 of the EPROM area contains a utility package which is used by the operating system in order to provide the features of the EPROM option. This area of code starts at 8000H and entry to this utility is via a series of jump instructions, each with a specific function:

8000H JUMP 1 Generation Routine
8003H JUMP 2 Load Routine
8006H JUMP 3 Verification Routine
8009H JUMP 4 Checksum Routine
800CH JUMP 5 Block Move Routine

Normally the utility routines are entered from DEMOS by use of the extended DEMOS commands. Users who wish to utilise the facilities provided in machine code may make direct access to the routines by use of a small amount of machine code which will perform the paging of the system. The machine code must be contained in a common area of memory above 0C000H.

```
PAGE    EQU     0F7C6H
BPAGE   EQU     0E0H

ACCESS: LD      A,(PAGE)        ; get current page
        PUSH    AF
        LD      A,1
        OUT     (BPAGE),A       ; select ROM 1 page
        LD      (PAGE),A

ROUTINE: CALL   0H              ; call routine
        POP     AF
        OUT     (BPAGE),A       ; get return page
        LD      (PAGE),A
        RET
```

Generation Routine

This routine, when called, contains the program for the generation of the memory image from a program in the TPA.

Once called this program prompts the user to enter the title field and security code which is to be incorporated in the EPROM. It then generates the necessary checksums, interrupt vector field and various flags and prompts to output the EPROM image in Intel Hex form to suitable EPROM programmer If the program is large enough to require two 32K EPROMs, the program automatically will prompt to generate the second EPROM if required.

If a second EPROM is required, the user will be asked if an offset is required during the generation of this second EPROM. This is to accommodate different programmers, some of which may not accept Intel Hex code which has addresses of 8000H or greater If the question is answered with a 'Y' then an offset is added to the Intel Hex address generated which effectively changes addresses 8000H and above to addresses 0000H and above. The ELAN E2 programmer used by Husky Computers will accept addresses above 8000H and thus an 'N' answer will suffice (but remember to program the EPROM from address 8000H on the ELAN programmer).

Load Routine

This routine checks and loads the EPROM image into the TPA. (Equivalent to ROM n command). The accumulator on entry must contain the value 0 or 1 to define the EPROM image required.

Verification Routine

This routine returns the title field on the computer screen. (Equivalent to ROM VER n command.) The accumulator, on entry, must contain the page value to define the EPROM image required.

Checksum Routine

This routine calculates and checks the checksum of a EPROM image. On entry the accumulator must contain 0 or 1 to denote the image required. On exit, if the accumulator is zero, it signifies the checksum is valid. A non-zero value signifies an error.

Block Move Routine

This routine moves an area of EPROM into a specified area of the TPA. On entry the DE register pair must point to a parameter block which contains the transfer parameters:

DE --->	1	Source Page (1 byte)
	2	Source Address (2 bytes)
	2	Destination Address (2 bytes)
	2	Block length (2 bytes)

This routine simply provides a block move without checksum or verification.

For more details of load programs see Technical Note TN243.

13.9.8 Important Note

The supply and installation of user EPROMs is the customer's responsibility, however, it is advised that this is carried out by Husky Computers during build time prior to shipment, so that system tests can be performed.

Chapter 14 Using Escape Codes (9G08 only)

14.1 Introduction

The facilities described in this chapter are not available in versions earlier than 9G08.

A problem encountered with earlier versions of the operating system was that any machine code that used non-CP/M routines within the operating system, invariably needed changing to run correctly on later versions, since the operating system entry points often move to different addresses in memory.

For this reason, the 9G08 operating system includes an escape sequence handler which allows access to routines within the operating system by a version-independent means.

These escape sequences are strings of characters which are sent to the console using BDOS function calls 2 or 9, each string beginning with the escape code (1BH, 27).

BDOS function call 9 must be used with care since any data equal to 24H (36 decimal) will cause output to cease, leading to unpredictable results.

The following example illustrates the use of the escape sequence to turn the cursor off. Note that the example contains its own display routines so that the message end can be specified by a character that will not be found within the message.

```
; demonstration of escape sequence to turn cursor off
                org     0100h

bdos:           equ     5h          ;bdos call location
esc:            equ     1bh         ;escape code
gr:             equ     80h         ;graphics identifier
cursor:         equ     1           ;cursor state identifier
on:             equ     0           ;cursor on code
off:            equ     1           ;cursor off code
strend:         equ     0ffh        ;end of message marker

start:          ld      hl,escmes   ;set pointer to string
                call    display                 ;output to console
                ret
```



```
display:        ld      a,(de)      ;fetch character
                cp      strend      ;is it string end
                ret     z           ;if yes return
                push    de          ;else save pointer
                ld      a,e         ;put char in e reg
                ld      c,2         ;set for console output
                call    bdos        ;write char to screen
                pop     de          ;restore pointer
                inc     de          ;increment the pointer
                jr      display     ;loop for next

escmes:         dc      esc,gr,cursor,off   ;message for cursor off
                end
```

This example can form the core of a graphics package and has the advantage that programs will be smaller and will not need updating to run with future versions of the operating system.

The escape sequences available in 9G08 will now be described.

The user program is responsible for checking the range of values passed to these routines.

14.2 LINE: draw lines or boxes

Sequence: ESC 80H 0 P1 P2 P3 P4 P5 P6 P7 P8 P9
80H: Graphics Identifier
0: LINE ID byte
P1. Start co-ordinate x1 lsb
P2: Start co-ordinate x1 msb
P3: Start co-ordinate y1
P4: End co-ordinate x2 lsb
P5: End co-ordinate x2 msb ie 16 bit value
P6: End co-ordinate y2 ie 8 bit value
P7· Colour: = 0 White line = 1 Black line
P8: Box: = 0 Line = 1 Box
P9· Fill: = 0 Empty box = 1 Fill box

14.2.1 Remarks

1) The fill parameter has no effect if P8 = 0
2) 0< =x< =239 and 0< =y< =63

14.3 CURON/CUROFF: turn cursor on/off

Sequence: ESC 80H 1 P1
80H. Graphics Identifier
1: Curon/curoff ID byte
P1. 0 turns the cursor on, 1 turns the cursor off

14.3.1 Remarks

1) Applicable to both text and graphics modes.

14.4 POINT: read pixel colour

Sequence: ESC 80H 2 P1 P2 P3

80H. Graphics Identifier
2: POINT ID byte
P1. x co-ordinate lsb
P2: x co-ordinate msb
P3: y co-ordinate

14.4.1 Remarks

1) On return A = 0 if pixel is white, 1 if pixel is black, -1 if co-ordinates are out of range
2) 0 < = x < = 239 and 0 < = y < = 63

14.5 LOCATE: move cursor and turn it on/off

Sequence: ESC 80H 3 P1 P2 P3 P4
80H. Graphic Identifier
3: LOCATE ID byte
P1. x co-ordinate lsb
P2: x co-ordinate msb
P3: y co-ordinate
P4: Cursor on or off: 0 = on 1 = off

14.5.1 Remarks

1) Text Mode: 1 < = x < = 80 and 1 < = y < = 23
2) Graphics Mode:
   CHAR0: 0 < = x < = 239 and 0 < = y < = 63
   CHAR1. 0 < = x < = 30 and 0 < = y < = 5
   CHAR2: 0 < = x < = 30 and 0 < = y < = 5
   CHAR3: 0 < = x < = 30 and 0 < = y < = 5
   CHAR4: 0 < = x < = 30 and 0 < = y < = 5

14.6 CHAR: character set and video attributes

Sequence:	ESC 80H 4 P1 P2
80H:	Graphic Identifier
4:	CHAR ID byte
P1.	0-4 specifying character size 0-4
P2:	Video attribute: 0 is Dark char/light background, 1 is Light char/dark background

14.7 SCREEN: select graphics or text mode

Sequence:	ESC 80H 5 P1
80H.	Graphic Identifier
5:	SCREEN ID byte
P1.	0 is Text mode, 1 is Graphics mode

14.8 PSET: set a pixel to black

Sequence:	ESC 80H 6 P1 P2 P3
80H.	Graphic Identifier
6:	PSET ID byte
P1:	x co-ordinate lsb
P2:	x co-ordinate msb
P3:	y co-ordinate

14.8.1 Remarks

1) 0 < = x < = 239 and 0 < = y < = 63
2) Only ONE pixel is set (to black)

14.9 PRESET: reset a pixel to white

Sequence:	ESC 80H 7 P1 P2 P3
80H.	Graphic Identifier
7:	PRESET ID byte
P1.	x co-ordinate lsb
P2:	x co-ordinate msb
P3:	y co-ordinate

14.9.1 Remarks

1) 0 < = x < = 239 and 0 < = y < = 63
2) Only ONE pixel is reset (to white)

14.10 CIRCLE: draw a circle

Sequence:	ESC 80H 8 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14
80H.	Graphic Identifier
8:	CIRCLE ID byte
P1.	Centre x co-ordinate lsb
P2:	Centre x co-ordinate msb
P3:	Centre y co-ordinate
P4:	Radius lsb
P5:	Radius msb
P6:	Colour· 0 = White, 1 = Black
P7 P14:	Filler bytes, usually filled with the following sequence: 0 0 0 0 1 1 0 0

14.10.1 Remarks

1) 0 < = x < = 239 and 0 < = y < = 63

14.11 Caps Lock: set or reset caps lock mode

Sequence:	ESC 85H P1
85H.	Caps Lock Identifier
P1.	0 = turn Caps lock off, 1 = turn Caps lock on

14.12 Set Viewing Angle

Sequence: ESC 87H P1

87H: Set Viewing Angle Identifier

P1. 0 - 15 (decimal): Degree of shade

14.13 Backlight: turn backlight on or off

Sequence: ESC 8AH P1

8AH: Backlight Identifier

P1. 0 = turn off, 1 = turn on

14.14 Set Security Escape Code

Sequence: ESC 8BH P1 P2 P3 P4 P5

8BH: Set security escape sequence identifier

P1-P5: Five ASCII characters forming the security escape sequence placed into the system escape buffer.

14.15 Default Security Escape Code: set escape code to default

Sequence: ESC 8CH

8CH: Default security escape code sequence identifier

14.15.1 Remarks

Loads the default security escape sequence of 56580 into the system escape buffer

14.16 INS Key: restore window position

Sequence: ESC B9H

B9H. INS identifier

14.16.1 Remarks

Restores screen to the original position before any of the looking keys were used.

14.17 Audible Output Control: turn sound output on or off

Sequence: ESC BAH P1

BAH. Audible output control identifier

P1. 0 enables audible output, 1 disables audible output

HUSKY HUNTER 2 TM

Part 5
Hardware

Chapter 15 Interfacing

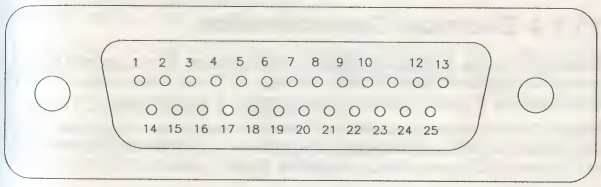
15.1 RS232 Interface (Communications Port)

15.1.1 Introduction

Hunter 2 uses a standard communications line via a 25 pin "D" connector compatible with the requirements of EIA specification RS-232-C. The connections are summarised in section 15.1.3.

Hunter 2 is configured as a 'DTE' (Data Terminal Equipment).

15.1.2 Connector Layout



15.1.3 Pinouts

Pin	Description	Pin	Description
1	Protective Ground	14	(Not used)
2	Transmitted Data	15	Transmit Clock
3	Received Data	16	(Not used)
4	Request to Send	17	Receive Clock
5	Clear to Send	18	(Not used)
6	Data Set Ready	19	(Not used)
7	Signal Ground	20	Data Terminal Ready
8	Data Carrier Detect	21	(Not used)
9	(Not used)	22	Ring Indicator
10	5v Power Out	23	(Not used)
11	(Not used)	24	(Not used)
12	(Not used)	25	Charger
13	(Not used)		

Note that Pins 1 and 7 are connected together and to the case.

15.1.4 Electrical Characteristics

Hunter 2's output voltages on signals "from **Hunter 2**" are guaranteed to be greater in magnitude then 5V when driving a 4K7 load, in accordance with EIA specifications. Typically output voltages are 8V in magnitude.

Hunter 2 detects input signals with a "fail safe" characteristic. Inputs have a logic 1 threshold of > 3.5V and a logic 0 threshold of < 1.5V. Handshake inputs (such as CTS and DCD) are pulled to an active (logic 1) state.

To summarise:

Output:	
Logical "1" or marking Logical "0" or spacing	Output > +5V with 4K7 load. Output < -5V with 4K7 load.
Input:	
Logical "1" or marking Logical "0" or spacing	Input > +3.5V Input < +1.5V

Each input provides a line terminator impedance of 6K8.

15.1.5 Interface Power Control

Under normal circumstances the serial output interface is powered down to conserve battery life. The first character which requires transmission causes the interface to be powered up. A pause of approximately 2 seconds is allowed for the interface to settle down and then the character is sent. Further characters are not affected in this way Once powered the interface is latched on until **Hunter 2** is switched off or power removed using an OUT instruction in either BASIC or machine code.

When the interface is powered up it may cause spurious characters to appear to be sent out, causing problems with the receiving system. These may be overcome by powering the interface, prior to requiring the data, by sending any character as soon as **Hunter 2** is switched on. A BASIC LPRINT command will achieve this.

Alternatively, the interface may be controlled as follows:

OUT 132,1 will turn the interface on
OUT 132,0 will turn the interface off

If the interface is turned off after characters have been sent, then further characters will not cause the interface to power up and they will be lost.

15.1.6 Connector Locking

Two locking tabs are provided, to allow you to secure the mating connector The recommended clips are: ITT Canon D53018.

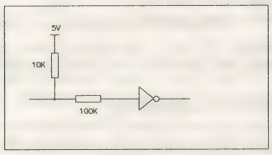
15.2 Single Bit Input Port (LEMO)

15.2.1 Introduction

The single input bit has been implemented for use with devices such as barcode wands, voltage free switch contacts, etc. It is provided via the 4 pin LEMO socket pin 2. Note should be made that it is shared with the TXCLK on the RS232 interface (pin 15). It should not, therefore, be used at the same time as the communications. (Pin 3 (charger) is also shared with pin 25 on the D-type connector, to provide an alternative charging connection.)

15.2.2 Circuit Implementation

The input is configured as a protected, pulled up CMOS compatible line:

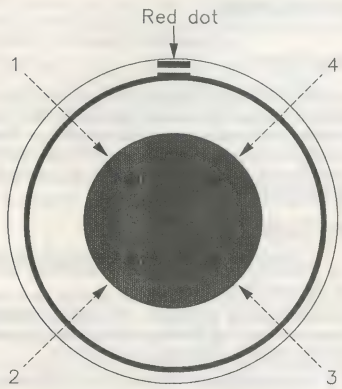


It has a logic 1 threshold of > 3.5V and a logic 0 < 1.5V It is protected to ±25V

The pull-up resistor enables the use of a switch to ground to provide the input.

No debounce circuitry is provided. If it is required then software must be written for this purpose.

This is bit 3 on I/O port 2.



Pin	Description
1	GROUND
2	TXCLK or input
3	Charger
4	5V output (Wand option only)

15.2.3 The LEMO Connector

To connect the LEMO plug, align the red mark on the plug sleeve with the corresponding red mark on **Hunter 2**'s LEMO socket and align the square pin on the plug body with the key-way on the socket. Push the LEMO plug firmly into **Hunter 2**'s socket until the outer sleeve of the plug is flush with the casing of **Hunter 2**'s LEMO socket.

To remove the LEMO plug, pull back the spring-loaded outer sleeve then pull the plug out of **Hunter 2**'s socket.

Chapter 16 Batteries

16.1 Introduction

Hunter 2's power control system is designed to optimise battery life by adapting power consumption to the level of activity. This means that power consumption varies according to the level and nature of use, making calculations of battery life slightly more complex than usual. However, if it is known how long **Hunter 2** will be used in its different modes, it is possible to calculate with reasonable accuracy the expected life of a set of batteries.

16.2 Battery Capacity

Battery capacity is measured in mAH (milliamp-hours) and by knowing the battery type in use, life may be calculated.

Battery capacity is affected by a number of factors including age, charging method, ambient temperature, etc. However, the capacity typically available from the two types of battery used by **Hunter 2** is as follows:

* DISPOSABLE Duracell type MN1500 (I.E.C. LR6) capacity 900 mAH.
* RECHARGEABLE Berec type NCC50 (I.E.C. KR 15/51) capacity 425 mAH.

Because capacity can vary significantly, the application planning should always assume reduced available capacity There are no firm guidelines for this, but 50% of capacity is considered conservative.

A factor affecting apparent battery life is the "end point", or voltage at which the battery is considered to be exhausted. **Hunter 2** sets this end point at 4.6 - 4.8 volts for battery warning.

This value yields virtually 100% of the Nickel Cadmium batteries' capacity and gives good results for Alkaline cells, but means that manufacturer's specifications for capacity may sometimes be misleading.

16.3 Power Modes

While turned on, **Hunter 2**'s power consumption has three principal modes:

* POWER SAVE (awaiting keyboard or serial port entry): 15mA
* OPERATING (running a program for calculation, data storage, screen updating or any operation involving continuous execution of user program): 80mA (80K RAM) or 90mA (208K RAM).
* COMMUNICATING (connected to a fully equipped RS232 port and operating): 115mA (80K RAM) or 125mA (208K RAM)

As can be seen, the POWER SAVE current is very low Although the screen is on and the keyboard responsive to any key depression, this is the condition in which **Hunter 2** is expected to be most of the time.

A typical application example might be a salesman using **Hunter 2** for order entry purposes at a number of customer's premises and returning to base for communication.

16.4 Example Power Consumption

The day is 7.5 hours long, during which he makes five calls. At each call he uses **Hunter 2** for 40 minutes for stock checking and order entry purposes. The communication time takes 15 minutes, with a further 5 minutes left powered up.

Of the 40 minutes use at each call, only 5 minutes is considered 'operation' since most of the time **Hunter 2** is waiting for keyboard entry.

16.4.1 Each Visit

Operating Power 80 x 5 / 60 = 6.67 mAH

Power Save Power 15 x 35 / 60 = 8.75 mAH

Total Visit Power = 15.4 mAH

Total Day's Power:

5 visits at 15.4mAH = 77 mAH

Communicating 115 x 15 / 60 = 28.75 mAH

Waiting after comms 40 x 5 / 60 = 3.33 mAH

Day Total = 109 mAH

Total Time in use through the day: 3 hours 40 minutes.

It would be expected that the **Hunter 2** would last approximately 4 days (or 14 hours 40 minutes total) on a set of rechargeable cells. Alternatively, **Hunter 2** would last 10 days (or 36 hours 40 minutes) on a set of manganese alkaline cells.

16.4.2 Battery Life

In normal applications, the periods spent in power save mode, or in data retention mode (ie powered off) are much longer than the periods spent powered-up and processing.

16.5 Data Storage Life

This section clarifies the conditions which affect the quiescent battery life of the **Hunter 2** ie when it has been left switched off, but retaining data and program within its CMOS RAM.

Quiescent battery life is dependant on these factors:

* Quiescent current of the **Hunter 2**
* The remaining capacity of the batteries
* Temperature variation
* Type of battery installed

The **Hunter 2** utilises a considerable amount of CMOS technology. CMOS static RAM has the feature that, when it is quiescent (just retaining data) power supply current falls to a very low level. While it is difficult to quantify for individual memory chips an average of a number of chips (up to 104) can be assessed.

The average quiescent current is also temperature dependent and it should be noted that the **Hunter 2** is only tested for this parameter at normal room temperature. If temperature extremes are likely, the figures presented may well be misleading.

All **Hunter 2**s are tested for their quiescent battery current to be less than 0.1mA (typically, it is about 60 microamps). If a new set of manganese alkaline batteries are installed prior to storage, one would expect them to realise their maximum capacity (approximately 1,800 mA hours) yielding a life expectancy of 18,000 hours or approximately 750 days. This does not allow for the self discharge characteristic of the manganese alkaline cells which amounts to 10% capacity loss per year. Therefore, one would expect self discharge capacity loss to reduce the quiescent battery life to approximately 600 days.

The worst case for alkaline manganese cells is if the unit has been used until low battery warnings are being displayed. In this condition the battery has approximately one third of its capacity remaining, all of which is available for maintaining the memory during quiescent conditions, since the voltage required is low. Simple arithmetic, therefore, suggests 200 days remain after battery warning on low manganese alkaline cells.

Nickel-Cadmium (NiCad) batteries used fully charged have a capacity of 500mA hours which again would suggest a quiescent battery life of approximately 200 days. However, at high temperatures, NiCads self-discharge quite rapidly. For extended shelf life it is not recommended to leave NiCads installed in the **Hunter 2**.

If the main batteries are removed, data storage relies on the internal NiCad battery. This has a capacity of 90mA hours and under normal conditions will remain fully charged by a trickle current from the main batteries. This would suggest 900 hours quiescent battery life under normal temperature conditions, or one month's shelf life.

It is necessary to emphasise that all the above notes relate to a **Hunter 2** stored at normal room temperature. At 50 degrees Centigrade, quiescent current increases quite dramatically (by a factor of approximately five times). It is not recommended, therefore, to store the **Hunter 2** under these conditions.

16.6 Mains (Line) Powered Installations

Although primarily designed for portable applications, **Hunter 2** may also be used in a fixed location. If continuous operation is required, an external power supply is often called for There are two possible approaches: permanent charging and mains power supply.

16.6.1 Permanent Charging

All **Hunter 2s**, with the exception of the Intrinsically Safe versions, can be fitted with rechargeable cells. For permanent installations it is permissible to leave the cells permanently on charge (only the correct Husky charger may be used).

Advantages

* No danger of damage to the **Hunter 2**.
* Automatic battery backed operation in case of mains supply failure.
* No special hardware required.
* The **Hunter 2** can at any time be unplugged and used in a portable mode.

Disadvantages

In comparison with the usual cyclic charging procedure, the following problems may be experienced:

* Rechargeable cells experience "Memory Effect" If a cell is permanently topped up and never discharged, it "forgets" its capacity to provide charge and consequently will appear to go flat very quickly the first time the charging is discontinued. The duration of mains failure that can be tolerated may, therefore, be considerably less than expected. Full capacity can be regained by several deep discharge cycles. In critical schemes requiring operation after mains failure, a regular full discharge should be forced to minimise the memory effect. A period of 1 to 3 months between forced discharges is suggested.
* Continual charging at the full rate can cause drying of the electrolyte and premature failure of the cells, with a reduction in capacity after a shorter than usual period of time. It must be realised that Nickel Cadmium Cells have a limited life anyway, but that this is reduced by continual charging. Three ways of coping with this problem are:
    * Replace the cells regularly (eg annually)
    * Give the cells a regular capacity test by way of charge/timed discharge cycles.
    * Use a special low current trickle charger.
* The **Hunter 2** has a fairly small battery and consequent charging current. This current may be inadequate to maintain full operation, especially if communications or a barcode wand are in continuous use. Under these circumstances, the battery may go flat, causing the **Hunter 2** to switch off. The charging current will be adequate if keyboard entries are involved, and the **Hunter 2** is allowed to enter "power save" between keystrokes.
* Further reduction in battery life will result if the battery is allowed to heat up. Check that ventilation is adequate for both **Hunter 2** and charger Temperatures of 5 to 25 degrees C are ideal.

Manufacturers of NiCad batteries recommend that these cells are not charged below 0 degrees C. Note that deterioration of the battery is likely if it is continually charged at the full rate at sub zero temperatures. In this case, a special low current trickle charger is required, but a situation can be reached where the charging current cannot keep up with the power consumption of the **Hunter 2**. This is not easily defined, since it varies with the way the **Hunter 2** is used as well as the model.

16.6.2 Mains Power Supply

This method involves removing the battery altogether and using a regulated mains power supply or external battery supply to supply power to the **Hunter 2**.

Advantages

* No maintenance.
* Temperature-independent.

Disadvantages

* A badly designed or faulty supply unit may destroy the **Hunter 2**.
* The **Hunter 2** could be susceptible to noise travelling along the mains supply.
* A special regulated power supply is required, capable of providing 6V + 5% at 250mA continuous (typical consumption is considerably less).
* Loss of mains causes instant loss of operation. Program and data are retained by a back-up battery, but this is not normally guaranteed for more than 200 hours.
* If a battery is accidentally fitted, it may leak chemicals or hazardous gases, or explode and damage the **Hunter 2**.

16.6.3 Conclusions

For continuous operation, it is suggested that NiCads be fitted with a permanent charging arrangement. This is strictly forbidden for Intrinsically Safe units, unless their safety features are not wanted, as NiCad are not rated as Intrinsically Safe.

Permanent charging of **Hunter 2** is acceptable, but under conditions of heavy use, such as continual use of RS232 communications or a barcode wand, 100% operation cannot be guaranteed.

16.6.4 Technical Details

Technical details of mains power supply and charger design are available as Technical Note number 177, on request from Husky Computers.

HUSKY HUNTER 2™

Part 6
Appendices

Appendix I Hunter 2 Specification

Physical

Construction	Die-cast aluminium alloy.
Size	216mm x 156mm x 32mm.
Weight	(Including batteries) 1150 grams.
Sealing	Waterproof against accidental immersion.
Straps	Wrist strap on either side.

Fascia

Screen	240 x 64 dot full graphic liquid crystal display with keyboard controlled contrast adjustment.
Keyboard	58 keys arranged in four rows of 15 QWERTY layout with function and control keys. Fully waterproof sealed rubber keys.

Programming

Built-in language	Extended BASIC Interpreter.
File handler	CP/M compatible software environment allows loading of many popular programs.
Storage	Programs and data stored in battery supported memory Operating system stored in firmware.
Screen handling (text mode)	The actual screen of eight 40 character lines forms a window on a larger virtual screen. The window is controlled with the cursor keys.
Screen handling (graphics mode)	Both text and graphics information are displayed together. There are five selectable character sizes giving facilities for elegant, professional screen formatting.
Keyboard	All keys reprogrammable in software.

Communications

Type	RS-232/V24 serial port on standard 25 pin 'D' type connector
Configuration	Entirely software controlled.
Baud rate	Up to 4800 baud asynchronous.
Protocols	Standard 'invisible' protocols for flow control and security are transparent to user programs. Formats provided include: (a) none/simple 'TTY' communication (b) XON/XOFF for mini and main frame computer systems (c) ETX/ACK for many popular printers (d) ACK/NAK for secure telephone communication (e) SYSTIME, with BCC error checking (f) 2780 fully synchronous implementation (optional)
Handshaking	RTS, CTS, DTR, DSR, RI, CD, all selectable by software.

Graphics

Resolution	240 x 64 dots
Commands	Software support for lines, boxes, circles and points.

Sound

Four octave range, fully programmable for note and duration.

Character Sets in Graphics Mode

Five different sizes, selectable from BASIC and CP/M

True descenders in larger sizes.

Reverse video in all character sets.

Memory

Type	CMOS low power semiconductor RAM.
Retention	Battery support with soldered in NiCad backup.
Capacity	Available in various RAM size options.
Firmware	Built-in operating system including File Manager, BASIC Interpreter and disk emulator
Upgrading	RAM memory can be upgraded on return to factory to larger size

Real Time Clock

Type	Software accessible time of day and calendar clock. Totally independent of microprocessor.
Accuracy	200ppm worst case, typically 2 sec per day.
Integrity	Non volatile, battery backed.

Microprocessor

Type	NSC800-4
Software	Executes the Z-80 instruction set (8080 super set); can run standard CP/M compatible programs.
Cycle time	250ns (4MHz)

Batteries

Main batteries	Four 'AA' cells, NiCad rechargeable for daily use. Mallory MN1500 for long shelf life and reliability. Typical operating lifetime (software dependant) 14h NiCad 45h alkaline.
Secondary	Built-in NiCad charged from main cells when operating. Life 20h when main batteries are discharged.

Husky Computers reserve the right to alter the specification or conditions of any product or service without prior notice.

Appendix II Operating Environment

Operating Temperature

The recommended operating temperature is from 0 C to 50 C. Below 0 C the response time of the screen increases and below -5 C the screen is too slow for practical use.

Above 62 C the screen will not operate. Permanent damage may occur if exposure is prolonged. Program execution and data retention are expected to remain normal in the range of -20 C to +50 C, but are not guaranteed.

Humidity

Hunter 2 is designed to operate in conditions up to 100% relative humidity The unit has an internal desiccator and a visual indicator shows any water ingress.

Waterproofing and Dustproofing

Hunter 2 is designed to resist the ingress of water and dust and generally meets the requirements of IP65. Factory pressure testing to 5 psi internal pressure assures that gross leaks do not occur and the unit will generally withstand brief (less than 30 sec) immersion in shallow (not more than 12 inches) water depth, although some water ingress may occur. Immersion characteristics are not guaranteed and can be impaired by maintenance, impact damage or modification. The commonest cause of failure is loose battery caps.

Trapped water in the keyboard area can sometimes lead to progressive keyboard failure, exhibited by 'over sensitive' or 'latched on' keys. To avoid this, the unit should be allowed to drain by storing vertically in a warm place. Forced drying by devices such as hair dryers is not recommended.

Fine dust may interfere with keyboard operation by jamming keys. If possible, it should be blown out by compressed air.

Some liquids, especially coffee, can congeal in the keyboard and prove difficult to remove. In such cases, the unit should be rinsed carefully under a warm tap, having first checked the security of the battery cap.

Water can be trapped in the RS232 connector and can lead to galvanic corrosion of the gold plated pins due to porosity in the plating. This can occur even if the RS232 port is not energised. To avoid this problem, simply blow any trapped water when convenient.

Storage Conditions

Hunter 2 must be fitted with batteries. For long-term storage it is essential to fit a new set of alkaline cells, rather than rechargeable NiCad cells (NiCad cells often have a shelf life of only 3 months) Storage temperature range is -20 C to 60 C, but as battery life is shortened at higher temperatures we recommend storage at less than 35 C. Under these conditions we recommend batteries are replaced at least every 12 months. Data retention is only guaranteed in the range of 0 C to + 50 C.

A serious risk with long term storage is battery leakage, which can cause very serious damage to **Hunter 2**. Use only the highest quality Alkaline or NiCad cells for storage and ensure they are brand new before installation.

Electromagnetic Compatibility

Hunter 2 complies with BS 3G100 for electromagnetic compatibility, provided it is not modified. This assures suitability for use in aircraft and other applications where HF, VHF and UHF communications are used. The certification only applies to battery operation - problems may be experienced if external power is used.

Hunter 2 is also very tolerant of external electromagnetic interference and will not exhibit problems in the vicinity of electric transmission lines, trains or low power transmitters, although performance can be compromised by external connections.

Resistance to Shock, Vibration and Impact

Hunter 2 is designed to withstand the rigours of normal use. **Hunter 2**s are used in many demanding applications such as:

* Recording data by forestry workers
* Field use by armed services
* Meter reading by water authorities
* Surveying road conditions by highway authorities
* Recording data on the effect of agricultural chemicals in the field
* Field use by Antarctic survey teams

The unit will withstand considerable deliberate abuse, but results are never guaranteed. A typical example is a fall from 3 to 6 feet onto a carpeted solid floor, where survival is expected. However, repeated impacts of this kind lead to progressive deterioration and failure of components by metal fatigue and other phenomena.

Impact with a hard surface like concrete invariably causes case damage including in severe case distortion and cracking. Even moderate impacts against solid surfaces (striking a wall when carried on a shoulder strap, for instance) can impair water resistance. A common impact problem is battery failure, caused by the separation of laser welds used in the construction of alkaline cells.

Hunter 2 has survived severe tests such as 26 drops from 2m, but suffered serious exterior damage.

Case replacement for **Hunter 2s** is a relatively simple and cheap procedure in the event of serious damage.

Appendix III ASCII Character Set

Hex/dec	Char	Hex/dec	Char	Hex/dec	Char	Hex/dec	Char
00/0	NULL	20/32	SPACE	40/64	@	60/96	\`
01/1	SOH	21/33	!	41/65	A	61/97	a
02/2	STX	22/34	"	42/66	B	62/98	b
03/3	ETX	23/35	#	43/67	C	63/99	c
04/4	EOT	24/36	$	44/68	D	64/100	d
05/5	ENQ	25/37	%	45/69	E	65/101	e
06/6	ACK	26/38	&	46/70	F	66/102	f
07/7	BEL	27/39	'	47/71	G	67/103	g
08/8	BS	28/40	(	48/72	H	68/104	h
09/9	HT	29/41	)	49/73	I	69/105	i
0A/10	LF	2A/42	*	4A/74	J	6A/106	j
0B/11	VT	2B/43	+	4B/75	K	6B/107	k
0C/12	FF	2C/44	,	4C/76	L	6C/108	l
0D/13	CR	2D/45	-	4D/77	M	6D/109	m
0E/14	SO	2E/46	.	4E/78	N	6E/110	n
0F/15	SI	2F/47	/	4F/79	O	6F/111	o
10/16	DLE	30/48	0	50/80	P	70/112	p
11/17	DC1	31/49	1	51/81	Q	71/113	q
12/18	DC2	32/50	2	52/82	R	72/114	r
13/19	DC3	33/51	3	53/83	S	73/115	s
14/20	DC4	34/52	4	54/84	T	74/116	t
15/21	NAK	35/53	5	55/85	U	75/117	u
16/22	SYN	36/54	6	56/86	V	76/118	v
17/23	ETB	37/55	7	57/87	W	77/119	w
18/24	CAN	38/56	8	58/88	X	78/120	x
19/25	EM	39/57	9	59/89	Y	79/121	y
1A/26	SUB	3A/58	:	5A/90	Z	7A/122	z
1B/27	ESC	3B/59	;	5B/91	[	7B/123	{
1C/28	FS	3C/60	<	5C/92	\	7C/124	|
1D/29	GS	3D/61	=	5D/93	]	7D/125	}
1E/30	RS	3E/62	>	5E/94	^	7E/126	~
1F/31	US	3F/63	?	5F/95	_	7F/127	DEL

Appendix IV ASCII to EBCDIC Conversion Table

When using EBCDIC communications **Hunter 2** provides ASCII to EBCDIC conversion as follows (all values are in Hexadecimal):

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
00	NULL	00	20	SPACE	40
01	SOH	01	21	!	5A
02	STX	02	22	"	7F
03	ETX	03	23	HEX	7B
04	EOT	37	24	$	5B
05	ENQ	2D	25	%	6C
06	ACK	2E	26	&	50
07	BEL	2F	27		7D
08	BS	16	28	(	4D
09	HT	05	29	)	5D
0A	LF	25	2A	*	5C
0B	VT	0B	2B	+	4E
0C	FF	0C	2C		6B
0D	CR	0D	2D	-	60
0E	SO	0E	2E	.	4B
0F	SI	0F	2F	/	61
10	DLE	10	30	0	F0
11	DC1	11	31	1	F1
12	DC2	12	32	2	F2
13	DC3	13	33	3	F3
14	DC4	14	34	4	F4
15	NAK	3D	35	5	F5
16	SYN	32	36	6	F6
17	ETB	26	37	7	F7
18	CAN	18	38	8	F8
19	EM	19	39	9	F9
1A	SUB	3F	3A	:	7A
1B	ESC	27	3B	;	5E
1C	FS	22	3C	<	4C
1D	GS	1D	3D	=	7E
1E	RS	1E	3E	>	6E
1F	US	1F	3F	?	6F

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
40	@	7C	60	\	79
41	A	C1	61	a	81
42	B	C2	62	b	82
43	C	C3	63	c	83
44	D	C4	64	d	84
45	E	C5	65	e	85
46	F	C6	66	f	86
47	G	C7	67	g	87
48	H	C8	68	h	88
49	I	C9	69	i	89
4A	J	D1	6A	j	91
4B	K	D2	6B	k	92
4C	L	D3	6C	l	93
4D	M	D4	6D	m	94
4E	N	D5	6E	n	95
4F	O	D6	6F	o	96
50	P	D7	70	p	97
51	Q	D8	71	q	98
52	R	D9	72	r	99
53	S	E2	73	s	A2
54	T	E3	74	t	A3
55	U	E4	75	u	A4
56	V	E5	76	v	A5
57	W	E6	77	w	A6
58	X	E7	78	x	A7
59	Y	E8	79	y	A8
5A	Z	E9	7A	z	A9
5B	[	00	7B	{	C0
5C	\	E0	7C	|	6A
5D	]	00	7D	}	D0
5E	^	5F	7E	~	A1
5F	_	6D	7F	DEL	07

Appendix V Hexadecimal to Decimal Conversion Table

DecHex	DecHex	DecHex	DecHex	DecHex	DecHex	DecHex	DecHex
0 00	32 20	64 40	96 60	128 80	160 A0	192 C0	224 E0
1 01	33 21	65 41	97 61	129 81	161 A1	193 C1	225 E1
2 02	34 22	66 42	98 62	130 82	162 A2	194 C2	226 E2
3 03	35 23	67 43	99 63	131 83	163 A3	195 C3	227 E3
4 04	36 24	68 44	100 64	132 84	164 A4	196 C4	228 E4
5 05	37 25	69 45	101 65	133 85	165 A5	197 C5	229 E5
6 06	38 26	70 46	102 66	134 86	166 A6	198 C6	230 E6
7 07	39 27	71 47	103 67	135 87	167 A7	199 C7	231 E7
8 08	40 28	72 48	104 68	136 88	168 A8	200 C8	232 E8
9 09	41 29	73 49	105 69	137 89	169 A9	201 C9	233 E9
10 0A	42 2A	74 4A	106 6A	138 8A	170 AA	202 CA	234 EA
11 0B	43 2B	75 4B	107 6B	139 8B	171 AB	203 CB	235 EB
12 0C	44 2C	76 4C	108 6C	140 8C	172 AC	204 CC	236 EC
13 0D	45 2D	77 4D	109 6D	141 8D	173 AD	205 CD	237 ED
14 0E	46 2E	78 4E	110 6E	142 8E	174 AE	206 CE	238 EE
15 0F	47 2F	79 4F	111 6F	143 8F	175 AF	207 CF	239 EF
16 10	48 30	80 50	112 70	144 90	176 B0	208 D0	240 F0
17 11	49 31	81 51	113 71	145 91	177 B1	209 D1	241 F1
18 12	50 32	82 52	114 72	146 92	178 B2	210 D2	242 F2
19 13	51 33	83 53	115 73	147 93	179 B3	211 D3	243 F3
20 14	52 34	84 54	116 74	148 94	180 B4	212 D4	244 F4
21 15	53 35	85 55	117 75	149 95	181 B5	213 D5	245 F5
22 16	54 36	86 56	118 76	150 96	182 B6	214 D6	246 F6
23 17	55 37	87 57	119 77	151 97	183 B7	215 D7	247 F7
24 18	56 38	88 58	120 78	152 98	184 B8	216 D8	248 F8
25 19	57 39	89 59	121 79	153 99	185 B9	217 D9	249 F9
26 1A	58 3A	90 5A	122 7A	154 9A	186 BA	218 DA	250 FA
27 1B	59 3B	91 5B	123 7B	155 9B	187 BB	219 DB	251 FB
28 1C	60 3C	92 5C	124 7C	156 9C	188 BC	220 DC	252 FC
29 1D	61 3D	93 5D	125 7D	157 9D	189 BD	221 DD	253 FD
30 1E	62 3E	94 5E	126 7E	158 9E	190 BE	222 DE	254 FE
31 1F	63 3F	95 5F	127 7F	159 9F	191 BF	223 DF	255 FF

Appendix VI Useful Utility Programs

Monitoring for low batteries in application programs

The operating system detects the low battery condition by checking bit 2 of port 2, which is the output of a voltage comparator that goes high when the battery power goes low. To check the level from a BASIC program, you can use the same technique.

For example, to check that there is sufficient power to use communications, you could call the following routine:

```
1000 OUT 132,1: REM POWER UP COMMS PORT
1010 J=INP(2): J=J AND 4
1020 K=INP(2): K=K AND 4
1025 OUT 132,0: REM POWER DOWN COMMS PORT
1030 IF (J AND D)=0 THEN RETURN
1040 CLS
1050 PRINT "PLEASE CHANGE MY BATTERIES"
1060 BEEP: BEEP: BEEP: RETURN
```

Note that this routine reads the level twice (in lines 1010 and 1020), then combines the readings (in line 1030) and only reports a low power condition if both readings agree. This helps make sure that momentary fluctuations in level are not reported as the low battery condition.

Restoring an accidentally deleted file

If you have deleted a file from the RAM disk, you can recover it as follows, providing that no other disk access have occurred since.

When you delete a file, its data is still intact on the disk, but the space used by it is marked as available in the FAT (file allocation table) and the first byte of each entry in the directory for this file is set to 0E5H.

To recover the file, select BASIC, type in and run the following program. **DO NOT SAVE IT TO DISK FIRST, OR YOU MAY OVERWRITE THE FILE YOU'RE TRYING TO RECOVER!**

```
10 CLS: PRINT "Enter Filename"
20 PRINT @ (4,2) "[AAAAAAAAXXX]"
30 INPUT @ (5,2)"", N$
40 DEFSEG = 1
50 FOR X=256 TO 2048 STEP 32
60 IF PEEK(X) = 229 THEN GOTO 1000
70 NEXT
80 DEFSEG = 0: PRINT "Completed": END

1000 Y$=""
1010 FOR Y=1 TO 11
1020 Y$=Y$+CHR$(PEEK(X+Y))
1030 NEXT
1040 IF N$=Y$ THEN POKE X,0
1050 GOTO 70
```

Run the program, and type the filename when prompted. Use capitals and type spaces to fill in the name and extension if they are shorter than the maximum.

Once the program has finished, return to the File Manager and use the DIR command to check that the file has been recovered. If it has, copy it to another machine using the SEND command, or to an Oracle disk. Check the copy, then delete the file from the **Hunter 2** disk.

If there are several files with the same name, there was more than one deleted version of the file. It is up to you to decide which one to use (for example, by checking the size or contents).

Identifying the operating system version from an application

Some application programs which require access to Operating System specific routines can be made to operate on various Operating System levels by first determining the version in use.

The following common RAM locations contain the indicated values for the three Operating Systems. This allows applications to determine the Operating System version, eg by using a PEEK command in BASIC.

Location	9G	9G+/9G06	9G08
E2C4H (58052)	7BH (123)	59H (89)	A8H (168)

Converting files to Intel Hex format

On the host computer, type the following program into a text file with the name UNLOAD.HEX. Before you can use UNLOAD, you must convert this file to a .COM file, using the following command:

LOAD UNLOAD.HEX (ENTER)

This will create a file with the name UNLOAD.COM that can be used to convert any file to INTEL HEX format. To use it, type the following:

UNLOAD *sourcefile filename*.HEX (ENTER)

where *sourcefile* is the name of the file to convert and *filename*.HEX is the name to use for the INTEL HEX version of *sourcefile*.

The INTEL HEX format file will always be located at 100H, but if all transferred data is saved into new files, (with the SAVE n command) then this is not a problem.

A source code version of UNLOAD is available on request from Husky Computers or authorised agents, either as a listing or 5.25" IBM compatible disk.

```
:10010000C32D023A3141303446363030453630303 6
:1001100032324545303132313041303133453330 80
:1001200030324344353430343231303330313336 89
:1001300033413233333363331323333363431323 183
:10014000354100 3A303030303030303030303030 B2
:100150001A000000000000000000000000000000 85
:10016000000000000000000000000000000000008F
:10017000000000000000000000000000000000007F
:10018000000000000000000000000000000000006F
:10019000000000000000000000000000000000005F
:1001A000000000000000000000000000000000004F
:1001B000000000000000000000000000000000003F
:1001C000000000014018000504F50202020202048C3
:1001D000455800008015 6D006E000000000000012
:1001E000000000000000150000000000000F50400
:1001F00044495343206572726F7224536F75726362
:10020000652066696C65206572726F7224436F6E3B
:10021000766572746965 67202E2E2E202424436F1B
:10022000 6D706C65746564202E2E2E20242A0600C5
:10023000F9CD5A04011000216C0011C601EDB02166
:100240006D007EFE20CA3904215D007EFE20CA3981
:1002500004115C000E0FCD05003CCC390411C60121
:100260000E13CD050011C6010E16CD05003CCC4B7A
:1002700004110D020E09CD0500111D020E09CD0558
:10028000001180000E1ACD0500115C000E14CD0582
:10029000000B7C40F03AF32C5013AC501FE8028E1A3
:1002A0005F3C32C501160021800019 7ECD990321E3
:1002B0000C0106003AC3014F097223730C0C79FE3E
:1002C00034280632C301C39902AF32C301CDC00343
:1002D000CDE50321C4017EFE012810360121030172
:1002E00011430101400 0EDB0C399023600210301 22
```



```
:1002F000118301014000EDB01143010E1ACD05003C
:10030000 11C6010E15CD0500B7C44B04C3990221D7
:10031000 0C0106003AC3014F09FE34280636302388
:10032000 3C1800CDC003CDE5033AC401F5FE012819
:1003300005114301180311830121030101400 0ED60
:10034000 80F1FE01280C3E0132C401210C013E0037
:10035000 18C71143010E1ACD050011C6010E15CDA7
:10036000 0500B7C44B04067F3E00214401772310EB
:10037000 FC060C2144013E30772310FC2150013E45
:10038000 1A7711C6010E15CD0500B7C44B0411C66E
:10039000 010E10CD0500C34204F5E60FFE0AFAA5D2
:1003A000 03C6371802F6305FF1CB1FCB1FCB1FCB34
:1003B000 1FE60FFE0AFABC03C63757C9F63057C905
:1003C0002AEE01111A001922EE01E57CCD99037A7B
:1003D000 3206017B320701E17DCD99037A320B01B3
:1003E0007B320901C9210401061E110000AF7E4FB6
:1003F000237E23FE40FAFA03D607E60F835F171722
:10040000 1717F57BE60F5F79FE40FA0F04D607E673
:10041000 0F8257F13E008A5710037BE60F5F7A17A1
:10042000 1717 17E6F0B3ED44CD99037A3240017BFC
:10043000 3241013E0D324201C911FB01CD5404C3CA
:10044000000 111E02CD5404C3000011F001CD5470
:10045000 4C300000E09CD0500C906EAAF2103015F
:10046000 772310FC111D02CD54042103 1363A23D9
:10047000 3631 23364121E60022EE01210A013E30C9
:10048000 7723 77C9E60F5F79FE40FA0F04D607E6B7
:10049000 F8257F13E008A5710037BE60F5F7A1721
:1004A000 1717 17E6F0B3ED44CD99037A3240017B7C
:1004B000 3241013E0D324201C911FB01CD5404C34A
:1004C000000 111E02CD5404C3000011F001CD54F0
:1004D00004C300000E09CD0500C906EAAF210301DF
:1004E000 772310FC111D02CD54042103 1363A2359
:1004F000 3631 23364121E60022EE01210A013E3049
:0A040800 1210A013E30772377C9FD
:00000001FF
```

Using the Oracle GT from Hunter BASIC

Introduction

The Oracle GT external disk drive can be used from the File Manager by using the DSK command, as described in Part 2. This section describes how to use the disk functions from applications written in Hunter BASIC, and provides guidelines on using them from assembly language and compiled languages.

These disk functions will execute from the BASIC transient program area. To provide space for them in the TPA, the BASIC MAXFILES statement is used.

It is important that you study this section carefully before attempting to use the interface in an application.

All the files described in this section are supplied on the Oracle GT system disk.

(If you require more details, including the source code and communications interface, refer to Husky Technical Note TN265.)

The ORACLE GT disk functions

The interface software provides six functions for controlling the Oracle GT·

a) Disk directory
b) File load
c) File dump
d) File erase
e) Rename file
f) Disk format

Functions (a), (b) and (c) support wild cards, so that groups of files may be copied from or written to the disk drive if required.

Reserving memory

MAXFILES is normally used in a BASIC program to specify the amount of memory to reserve for opening files. For each file specified by the MAXFILES statement, BASIC reserves 168 bytes from the transient program area (TPA).

MAXFILES can also be used to reserve memory for use by machine code, as described in Part 3 (section 10.11.3). The Oracle GT interface software is loaded into an area reserved using this technique, so you must execute the following statement in the application:

```
MAXFILES = 15
```

and the application must have no more than two files opened simultaneously, and **must** only use filenumbers #1 and #2 respectively (If you use filenumbers greater than #2 this will destroy the disk software.)

The machine code is loaded starting at address 0CE77H (52855D).

Entering the interface software

The BASIC statements required to create the interface machine code are stored on the system disk supplied with the Oracle GT, as POKE.BAS. Copy this file onto the RAM disk by using the DSK LOAD command (see Part 2).

You can then merge the statements into the BASIC application program. The program only needs to execute the statements once, as the machine code is stored above the normal program area.

Alternatively, you could execute the statements to create the machine code, then save the memory area as a file on the **Hunter 2** RAM disk. You could then use a loader program to copy this file to the storage area when required, if this is more convenient.

Passing parameters to/from the interface

Parameters must be passed between the Hunter BASIC and the disk utility software, as follows:

a) the required disk function number
b) the filename(s) where applicable
c) the status code variable
d) a selection code

Passing the status code

When a disk operation has been carried out and an error has occurred, an error code is returned to the calling program.

An application can access the status code by using the CALL function to call the disk routine:

```
ER = CALL(n)
```

where ER is the status code variable and *n* is the entry address to the disk routine (0CE77H, or 52855D).

Passing the disk function number

The disk function number is simply a number assigned to a disk function for the purpose of identification.

The assigned function numbers are as follows:

Disk Function	Function Number
Disk Directory	1
File Load	2
File Dump	3
Erase File	4
Rename File	5
Disk Format	6

The disk function number is passed to the disk software by using the Hunter BASIC ARG function, before calling the disk routine:

```
A = ARG(n)
```

where *n* is the disk function number required.

Passing filenames

Filenames are passed to the disk software via **Hunter 2**'s stack, by pushing a pointer to the string variable containing the filename onto the stack before calling the disk routine:

```
F$ = "FILENAMEEXT"
F = VARPTR (F$)
PUSH F
```

It is important to note that the filename must be in the correct CP/M format, as the disk software interprets the filename literally

The name must contain valid ASCII characters acceptable for a filename, and must be in upper case alpha characters.

On return from the disk software, the application must correct the stack by 'popping' the address off the stack, like this:

```
F = POP(1)
```

If not, the BASIC variable stack will accumulate addresses and eventually overflow, reporting a fatal stack error.

Passing the selection code

The disk FORMAT function requires a parameter to select the capacity of disk to format. This parameter is passed to the application via the BASIC ARG parameter, as the MSB of the argument.

For example to format a 720K capacity disk, you must set the selection code to '1', as follows:

```
A = ARG (6 + 256 * 1)
```

where the '1' selects 720K capacity and the '6' the format function

In order to format a 1.4M capacity disk the selection parameter should be '0'.

```
A = ARG (6 + 256 * 0)
```


Using the disk functions

The next six sections illustrate how to use each function by presenting simple BASIC programs. These programs are stored on the Oracle GT system disk.

Function 1: Obtaining a disk directory

DIRECT.BAS shows how to display the names of all files on the disk:

```
1 REM Demonstration of the DIRECTORY function
5 MAXFILES=15:GOSUB 10000
10 f$="???????????"
20 a=ARG(1):REM select directory of Oracle GT
30 f=VARPTR(f$)
40 PUSH f
45 CLS
50 PRINT"============* DIRECTORY *================"
60 er=CALL(52855)
70 PRINT"============* completed *================"
80 a=POP(1)
90 IF er <> 0 THEN BEEP: PRINT"Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

Note that the string variable f$ contains the character '?' in all locations of the name. This means that any filename will match with this name. A specific file or group of files may also be specified if required.

For example to obtain a directory of all .COM files on a disk the string variable f$ should be equated as follows:

```
f$ = "????????COM"
```

Function 2: Loading files from the Oracle GT

LOAD.BAS shows how to copy all .COM files from the Oracle GT disk to the Hunter 2 RAM disk:

```
1 REM Demonstration of the LOAD function
5 MAXFILES=15:GOSUB 10000
10 f$="????????COM"
20 a=ARG(2):REM select load from Oracle GT
30 f=VARPTR(f$)
40 PUSH f
45 CLS
50 PRINT"============* LOADING *================"
60 er=CALL(52855)
70 PRINT"============* completed *================"
80 a=POP(1)
90 IF er <> 0 THEN BEEP:PRINT "Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

Note that this function supports the use of wildcard characters. '?' is used in the string variable f$ to specify the group of files to copy

Function 3: Copying files to the Oracle GT

DUMP.BAS shows how to copy a group of files with the name DISK from the **Hunter 2** RAM disk to the Oracle GT disk:

```
1 REM Demonstration of the DUMP function
5 MAXFILES=15:GOSUB 10000
10 f$="DISK    ???"
20 a=ARG(3):REM select dump to Oracle GT
30 f=VARPTR(f$)
40 PUSH f
45 CLS
50 PRINT"-------------* COPYING *----------------"
60 er=CALL(52855)
70 PRINT"-------------* completed *----------------"
80 a=POP(1)
90 IF er <> 0 THEN BEEP:PRINT "Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

Function 4: Erasing files from the Oracle GT

ERASE.BAS shows how to delete the file DISCO.ASM from the Oracle GT disk:

```
1 REM Demonstration of the ERASE function
5 MAXFILES=15:GOSUB 10000
10 f$="DISCO   ASM"
20 a=ARG(4):REM select erase file from Oracle GT
30 f=VARPTR(f$)
40 PUSH f
45 CLS
50 PRINT"-----* ERASING: "+f$ + " *-----"
60 er=CALL(52855)
80 a=POP(1)
90 IF er <> 0 THEN BEEP:PRINT "Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

(To simplify the machine code, this function does not indicate an error if the file to be erased does not exist.)

Note that the erase function does **not** support the use of wild cards, so the filename must be specified exactly in the string variable.

Function 5: Renaming a file on the Oracle GT

RENAME.BAS shows how to rename the file OLDFILE.EXT on the Oracle GT to the name NEWFILE.EXT


```
1 REM Demonstration of the RENAME function
5 MAXFILES=15:GOSUB 10000
6 DIM f$(0,40)
10 f$="OLDFILE EXT"+"!"+"NEWFILE EXT"
20 a=ARG(5):REM select rename function
30 f=VARPTR(f$)
40 PUSH f
45 CLS
50 PRINT"-------------* RENAMING *----------------"
60 er=CALL(52855)
70 PRINT"-------------* completed *----------------"
80 a=POP(1)
90 IF er <> 0 THEN BEEP:PRINT "Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

Note how the two filenames are specified in the string variable f$, separated by the character '!' Wildcards are **not** supported by this function.

Function 6: Formatting a disk on the Oracle GT

FORMAT.BAS demonstrates how to format an Oracle GT disk, to 720K capacity.

```
1 REM Demonstration of the FORMAT function
10 MAXFILES=15:GOSUB 10000
20 a=ARG(6+256*1):REM select format of 720K capacity disk
45 CLS
50 PRINT"-------------* FORMATTING *----------------"
60 er=CALL(52855)
70 PRINT"-------------* completed *----------------"
90 IF er <> 0 THEN BEEP:PRINT "Error Code: ",er:END
100 PRINT "Operation completed without error":END
```

The Oracle GT can support two capacities of disk. To format the disk to the other capacity (1.4M), simply change the argument in the ARG statement as shown below:

```
20 a=ARG (6+256*0):REM select format of 1.4M capacity disk
```

Error handling

The status code variable is used by the disk software to report any error conditions that may arise during a disk operation to a controlling program.

The error message is returned onto the screen, and any progress information such as filenames being transferred parameters etc.

The error codes and their associated messages are shown below

00 - No error detected operation completed.

81 - No file

Search has failed to detect the file required.

B2 - Cannot open file

Access to open the source file has been refused. This is usually due to an error in the file itself.

B3 - Disk error

A timeout has occurred in the disk access. Check the connections are correct, a disk is in the drive or the baud rate is set to 4800 baud.

B4 - File read only

An attempt has been made to delete or write to a file with its read only attribute set.

B5 - Cannot make file

A failure has occurred in creating a destination file. There may be insufficient directory space, or the disk may be read-only.

B6 - Aborted

The operator has pressed CTL/FN C while a disk function is being executed.

B7 - Cannot write to file

A failure has occurred in writing to the file. There is probably insufficient space on the disk, or the disk may be read-only

B8 - Filename error

Wildcard characters ('?') have been used in the filename but are not allowed in this disk function.

B9 - Disk read only

The write protect toggle on the disk is set to read-only and an attempt has been made to write to the disk.

In addition to these software generated errors, Oracle GT disk errors may be returned in some instances. These additional errors which are generated by the FDC are listed below·

01H	Incorrect drive specifier
02H	Missing address marker
03H	Disk is write protected
04H	Record not found
08H	DMA over-run error
09H	Attempted DMA over 64K segment
10H	CRC error
20H	Floppy disk controller error
30H	Disk drive door status change
40H	Seek error
80H	Drive not ready

The following errors are generated by the controlling software in the disk drive itself:

A0H	Failed to format track
A1H	Directory space full
A2H	FAT space is full
A3H	File not found or failed to open for access
A4H	Read error
A5H	Write error
A6H	Close error
A7H	Read only access file

Use with machine code or compiled applications

The source file for the interface software is also supplied on the Oracle GT system disk, to allow you to assemble or link it with other languages.

For assembly applications there is no necessity for the code to be located in common memory, if required this could be relocated to a more convenient location in RAM.

Language information

The disk software is written in HD64180 assembly language. The development package used to produce the code was AVOCET AVMAC 180 CROSS-ASSEMBLER VER 2.00 running on an IBM AT

Using the disk functions

Depending on the application, not all of the disk functions may be required. It can be seen that the source file can be easily split to create a file with only those functions which are required. This file may then be linked into the main application source code or compiled code.

Accessing the RTC from applications

Turbo Pascal and Microsoft BASIC, like many other commercially available compiled languages, are targeted at a wide variety of machines, so they do not contain commands to carry out machine-specific tasks. It is therefore necessary to 'patch' the language with machine code routines to achieve machine-specific tasks such as accessing **Hunter 2**'s Real Time Clock (RTC).

The following machine code routine is designed to implement the four main functions required ie read and set the time, read and set the date:

```
clock:      equ     40h
clkday:     equ     0f7efh
clkmin:     equ     0f7ebh
years:      equ     0f7f3h

puttim:     call    bdos                    ;read the old values
            ld      b,4
            ld      de,63470
            call    put_1                   ;format time for write
write_clk:  xor     a                       ;get non test mode
            out     (clock),a
            out     (clock+14),a    ;stop the clock
            out     (clock+15),a    ;stop clock interrupts
            ld      hl,clkmin               ;start of regs
            ld      c,clock+4
            ld      b,8
clkslp:     ld      a,(hl)                  ;fetch the time function
            out     (c),a                   ;store to clock chip
            inc     hl
            inc     c
            ld      a,b
            cp      3                       ;last 2?
            jr      nz,chlpc
            inc     c                       ;skip over days of week
chlpc:      djnz    clkslp                  ;loop
```



```
            ld      a,(years)               ;fetch the years
            and     3
            inc     a
            ld      b,a
            ld      a,10h                   ;shift into ms half
yrsh:       rrca                            ;make into format
            djnz    yrsh
            out     (clock+13),a    ;use as leap year
            ld      a,1
            out     (clock+14),a    ;start the clock
            ret                             ;all done
gettim:     call    bdos                    ;read hardware clock into ram
            ld      b,4
            ld      de,63470
get_1:      inc     hl                      ;fetch pointer for write
            ld      a,(hl)
            inc     hl
            ld      h,(hl)
            ld      l,a
get_loop:   ld      a,(de)                  ;fetch b bytes from ram clock
            or      30h
            ld      (hl),a                  ;convert to ascii and store
            inc     hl
            dec     de
            djnz    get_loop
            ret
putdat:     call    bdos                    ;read old values of time
            ld      b,2
            ld      de,63472
            call    put_1                   ;format day for write
            ld      b,2
            ld      de,63474
            call    put_loop                ;format month for write
            ld      b,2
            ld      de,63476
            call    put_loop                ;write year to ram
            jp      write_clk               ;write day and month to hardware
put_1:      inc     hl
            ld      a,(hl)
            inc     hl
            ld      h,(hl)
            ld      l,a
put_loop:   ld      a,(hl)                  ;format b bytes for writing to h/w
            and     0fh
            ld      (de),a
            inc     hl
            dec     de
            djnz    put_loop
            ret
```

```
getdat:     call        bdos                    ;read hardware clock
            ld          b,2
            ld          de,63472
            call        get_1                   ;read day
            ld          b,2
            ld          de,63474
            call        get_loop                ;read month
            ld          b,2
            ld          de,63476
            jr          get_loop                ;read year
;function to read the realtime clock into memory
bdos:       push        hl
            ld          c,48
            call        5h
            pop         hl
            ret

            end
```

Implementing the routines in compiled MBASIC

Once the routines have been assembled using M80, the resulting relocatable file may be linked with a compiled MBASIC program. Each routine may be accessed in the following manner:

Time facilities

The routine to read the time requires a four character string to be provided for its output. The time will be returned in this string in the form "HHMM" For example:

```
1000 CALL GETTIM(HTIME$)
```

To set the time, set a four character string to the desired time (in the form "HHMM") and call the routine PUTTIM. For example:

```
2000 HTIME$ = "1200"
2001 CALL PUTTIM(HTIME$)
```

Date facilities

The routine to read the date requires a six character string for its output. The date will be returned in the form "DDMMYY". For example:

```
3000 CALL GETDAT(HDATE$)
```

To set the time, set a six character string to the desired date (in the form "DDMMYY") and call the routine PUTDAT For example:


```
4000 HDATE$ = "010187"
4001 CALL PUTDAT(HDATE$)
```

Please note that the correct length string must be passed to these routines, failure to do so will lead to unpredictable results.

Implementing the routines in TURBO PASCAL

The Turbo Pascal development system produces executable files without generating an intermediate, relocatable file which could then be linked to external routines such as machine code 'patches' You must therefore use the Turbo Pascal INLINE function.

To use INLINE, convert the Assembler routine to the corresponding machine code and then include it directly in the program source. For example, the following INLINE statements correspond to the GETTIM routine and will read the RTC, placing the resulting four character string in the variable TIME. (Note that you must define this variable to the correct length first.)

```
PROCEDURE GETTIM(VAR  TIME : STR4);
{GET TIME ROUTINE FOR TURBO PASCAL RUNNING ON A HUNTER}
BEGIN
  INLINE (14/48/205/5/0/6/4/33/>TIME+1/17/238/247/26/246/48);
  INLINE (119/35/27/16/248);
END;
```

To clarify the example, the following example Turbo Pascal program displays the time and waits for a key to be pressed:

```
PROGRAM REALTIME;
TYPE STR = STRING[4];
VAR
  TIME : STR4;
  A    : CHAR;
{$I GETTIM1.PAS}
BEGIN
  GETTIM(TIME);
  WRITELN('THE CURRENT TIME IS ',TIME);
  READ(KBD,A);
END.
```

Using alternate character sets in text mode

By default, **Hunter 2** displays only those characters which can be typed at the keyboard ie characters with ASCII values in the range 0-127 decimal. They comprise a full upper and lower case Roman alphabet, numerals and punctuation

symbols. By reprogramming the keyboard various combinations of different key values can be placed in the normal and shifted locations. This technique does have two limitations:

1 Only values in the range 0-127 may be placed into any key location.

2 The default key values are reset when you enter the system editor (EDIT) or perform a cold boot.

Hunter 2 can however display a much wider range of symbols and special shapes. If characters with ASCII values above 127 are sent to the screen controller then a completely different set of symbols are displayed. To see the full set available, execute the following simple program:

```
10 FOR X=128 TO 255
20 PRINT X;: OPCHR X
30 NEXT
```

You may display these characters with simple OPCHR statements or as a part of a print string. For example:

```
OPCHR 224,246,244,243,242,247 : REM Greek Alphabet
```

or

```
PRINT "De";CHR$(245);"tsche"  : REM German Umlaut
```

As mentioned earlier it is not possible to reprogram the keyboard with values greater then 127, but an application can intercept key presses and substitute another value before displaying it, if required.

For example, in PASCAL:

```
Function Getkey : byte;              (* Returns key value or *)
                                     (* Substituted code     *)
Var A : byte
Begin
   A:=BDOS(47,0)                     (* Read keyboard *)
   Case A of                         (* Convert if necessary *)
     97 : A :=225;                   (* Character 'a' *)
    111 : A :=239;                   (* Character 'o' *)
    117 : A :=245;                   (* Character 'u' *)
   End;
   Getkey:=A;
End;

Begin
  While true do write (chr(Getkey));
End.
```

The above program simply takes any key pressed on the keyboard and prints it except for the lower case letters a, o and u which are converted to their equivalents with Umlauts.

An alternative approach to this is to convert all screen output. This has the advantage that all program output, not just that entered by the user, is converted. Some high level languages such as TURBO PASCAL allow users to write their own I/O drivers. In other languages such as BASIC the technique is more cumbersome but just as effective.

The usual way to do this is to have a translation table - an array of character alternatives which are usually indexed by the ASCII value of the character to be converted:

```
0 DIM Alternative(128):REM PUT ALTERNATIVE CHARACTERS IN HERE
100 IF A>127 THEN PRINT CHR$(Alternative(ASC(A)-127));
    ELSE PRINT CHR$(A);
```

For a more detailed discussion of this topic, see Husky Technical Note TN268.

Appendix VII Intel Hex Format

Hunter 2 supports use of the industry standard 'Intel Hex' data format, to simplify transferring files to and from other computers. This involves converting the binary file to a text representation (INTEL HEX format), transferring that and then converting it back to binary on the receiving computer

Most CP/M computers provide an UNLOAD utility to convert binary files to INTEL HEX format and a LOAD utility to convert INTEL HEX to binary. **Hunter 2**'s file transfer commands (INP and SEND) provide automatic conversion as part of the transfer process (see Part 2 for details). For a listing of an UNLOAD utility, see Appendix VI.

The data is split into blocks with leader and trailer information and the data in between. Each data block contains the memory locations into which the data is to be placed.

A single block of data appears as:

:BBAAAAOODD.....DDCC

where: BB = No of data bytes
AAAA = Address of block
OO = Null record
DD = Data byte
CC = Checksum

Each byte is composed of two ASCII hexadecimal characters (0-9 and A-F). A block always contains 16 data bytes.

The record starts with a colon which is detected by the receiving system. This is followed by a count of the data bytes, the starting address, a delimiter, the data itself and a checksum on the end.

The checksum is calculated as the two's complement of the modulo 256 addition of all the preceding bytes.

The transmission ends with a block of zero length:

:0000000000

When loading, **Hunter 2** checks the address specified by each block to make sure that it will be loaded into a legal address in the TPA. **Hunter 2** also checks the checksum. If either check fails, an error message is displayed.

If the address is illegal:

```
* * Memory overflow * *
```


If the checksum indicates data corruption:

```
* * * Loading Error * * *
```

Reception of the end of record gives:

```
* * Loading completed * *
```

You may abort loading at any time by pressing ESC.

Appendix VIII Four Octave Sound Range

The following table indicates the pitch parameter to use in a SOUND command to produce a tone approximately the indicated frequency.

Note	Frequency (Hz)	Pitch parameter
C	138.810	493
D	146.830	439
E	164.810	391
F	174.610	368
G	196.000	326
A	220.000	290
B	246.940	260
C	261.630	244
D	293.660	217
E	329.630	193
F	349.230	182
G	329.000	162
A	440.000	144
B	493.880	128
Middle C	523.250	121
D	587.330	107
E	659.260	95
F	698.460	90
G	783.990	80
A	880.000	71
B	987.770	63
C	1046.500	59
D	1174.700	52
E	1318.500	46
F	1396.900	43
G	1568.000	38
A	1760.000	34
B	1975.000	30
C	2092.000	28

Appendix IX Graphics Routines

Introduction

This appendix describes machine code routines which provide the CP/M user with similar graphics facilities to those provided by BASIC.

NOTE: For users of 9G08 and later versions, the escape sequence mechanism described in Part 4 is preferable, as it involves less code and allows the program to be used with subsequent versions of the operating system without having to change system equates.

Program Description

Hunter 2 operates in two screen modes: a text mode which places the text characters on the screen, and a graphics mode which has five different character sets, normal and reverse video and a graphics facility for line and circle drawing.

The text mode is essentially a hardware feature, the character set being contained in ROM within the screen controller, making it faster than the graphics mode which is controlled and generated entirely by the operating system.

The text mode is defaulted to by the operating system, graphics being a feature of BASIC only

In order to gain access to the graphics facilities of **Hunter 2**, which are contained in the operating system EPROMs, links must be provided in order to handle the paging system of **Hunter 2**. The following routines provide these links, allowing machine code programs to utilise the graphics potential of **Hunter 2**.

As paging is involved, the following routines MUST be located in common RAM ie above 0C000H. A convenient area for locating this code is at USER.

A knowledge of machine code programming is assumed. The following is a description of each utility and the entry parameters which are required. To save program space, it is left to the user to ensure that the limits specified are not exceeded.

Graphics Functions

Clear the LCD Screen

Name: CLS
Entry: None
Purpose: The routine will clear the screen

Set LCD Screen to Graphics Mode

Name: GRAPH
Entry: None
Purpose: Switch the screen to graphics mode

Set LCD Screen to Text Mode

Name: TEXT
Entry: None
Purpose: Switch the screen to text mode

Switch Cursor on

Name: CURON
Entry: None
Purpose: Switch the cursor on

Switch Cursor off

Name: CUROFF
Entry: None
Purpose: Switch the cursor off

Select a Graphics Character Set

Name: CHAR
Entry: D reg = 0FFH for inverse video, D = 00H for normal video
E reg = 0,1,2,3 or 4 for particular character size
NOTE: Graphics mode must first be selected. This subroutine selects the character set required in graphics mode.

Set/Reset a Pixel

Name: PLOT
Entry: D reg = x co-ord. 0<x<0F0H
E reg = y co-ord. 0< =y< =03FH
A reg = 0 to clear a pixel, 1 to set the pixel
NOTE: Graphics mode must first be selected. This subroutine is to set or clear a given pixel.

Draw a Circle

Name: CIRCLE
Entry: E reg = radius required
H reg = x co-ord. 0<x<0F0H
L reg = y co-ord. 0< =y< =03FH
A reg = 0 to unplot a circle, 1 to plot a circle
NOTE: Graphics mode must first be selected. This routine will draw or erase a circle of a given radius about a given centre point.

Listing

The actual link routine required for the 9G operating system is as follows. Where values are different for 9G06, they are shown in brackets.

```
;RAM LOCATIONS UTILISED

PLTFLG      EQU         0FA4FH
GCURX       EQU         0FA4DH
GCURY       EQU         0FA4EH
ADDR        EQU         0FA53H
XPOS        EQU         0FA51H
YPOS        EQU         0FA52H
GCHRFLG     EQU         0FA55H
CHRCELL     EQU         0FA58H

;ENTRY POINTS TO THE ACTUAL OPERATING SYSTEM ROUTINES

GCLS        EQU         08C25H      (08C95H)
GMODE       EQU         08BFCH      (08C66H)
TMODE       EQU         08BBEH      (08C28H)
NCURON      EQU         094A6H      (094EFH)
NCUROFF     EQU         094BBH      (094D7H)
PUTCUR      EQU         090BEH      (09111H)
CURPXL      EQU         08C91H      (08D01H)
PRSET       EQU         08D6DH      (08DD1H)
ERACUR      EQU         090D6H      (09129H)

;THE FOLLOWING ROUTINES MUST BE LOCATED IN A COMMON AREA OF RAM AS
;THEY INVOLVE PAGING OF THE MEMORY. WHERE REGISTERS ARE REQUIRED TO
;BE SET, THE INFORMATION IS SUPPLIED. NO CHECKING IS MADE ON
;INFORMATION SUPPLIED TO THE ROUTINES, THIS IS LEFT TO THE
;RESPONSIBILITY OF THE USER TO ENSURE EXPECTED PARAMETERS ARE NOT
;EXCEEDED.

;THE STACK MUST BE LOCATED IN COMMON AREA, IE. ABOVE OC000H. EXECUTING
;CALLS TO GRAPHICS UTILITIES SUCH AS CHAR, CIRCLE ETC. WILL LEAD TO
;CORRUPTED SCREENS IF GRAPHICS MODE HAS NOT BEEN SELECTED FIRST

;THESE ROUTINES PROVIDE THE MAJORITY OF THE GRAPHIC ROUTINES NORMALLY
;ONLY AVAILABLE THROUGH BASIC.

;*********************************
;ROUTINE TO CLEAR THE LCD SCREEN
;*********************************

CLS:        RST         18H
            DC          B(GCLS)
```



```
;*****************************************
;ROUTINE TO SET LCD SCREEN TO GRAPHICS MODE
;*****************************************

GRAPH:      RST         18H
            DC          B(GMODE)

;*************************************
;ROUTINE TO SET LCD SCREEN TO TEXT MODE
;*************************************

TEXT:       RST         18H
            DC          B(TMODE)

;************************
;ROUTINE TO PUT CURSOR ON
;************************

CURON:      RST         18H
            DC          B(NCURON)

;*************************
;ROUTINE TO TURN CURSOR OFF
;*************************

CUROFF:     RST         18H
            DC          B(NCUROFF)

;*****************************************
;ROUTINE TO SELECT A GRAPHICS CHARACTER SET
;*****************************************

;N.B. THE CHARACTER SELECTION MUST ONLY BE MADE WHEN IN GRAPHICS
;MODE

; D REG = OFFH FOR INVERSE CHARACTERS
; D REG = 0 FOR NORMAL CHARACTERS
; E REG = 0,1,2,3,4 FOR GRAPHIC SETS 0,1,2,3 AND 4 RESPECTIVELY.

CHAR:       LD          A,(GCHRFLG)
            CP          E
            JR          Z,CHAR1
            PUSH        DE
            CALL        REMCUR          ;****** special utility ******
            POP         DE
            LD          A,E
            LD          (GCHRFLG),A
            XOR         A
            LD          (ADDR),A
            LD          (ADDR+1),A
            LD          (XPOS),A
            LD          (YPOS),A
CHAR1:      LD          A,D
            LD          (CHRCELL+27),A
            RST         18H
            DC          B(PUTCUR)
```

```
;************************************
;ROUTINE TO SET / RESET A GIVEN PIXEL
;************************************

;N.B. THIS COMMAND MUST ONLY BE MADE WHEN IN GRAPHICS MODE

; D REG = X CO-ORDINATE 0<X<0F0H
; E REG = Y CO-ORDINATE 0<=Y<=3FH
; A REG = 0 TO CLEAR THE PIXEL OR 1 TO SET THE PIXEL

PLOT:       LD          (PLTFLG),A
            LD          A,D
            LD          (GCURX),A
            LD          A,E
            LD          (GCURY),A
            RST         18H
            DC          B(CURPXL)

;************************
;ROUTINE TO DRAW A CIRCLE
;************************

;N.B. THIS COMMAND MUST ONLY BE MADE WHEN IN GRAPHICS MODE
;
; E REG = RADIUS REQUIRED
; H REG = X CO-ORDINATE OF THE CENTRE 0<X<0F0H
; L REG = Y CO-ORDINATE OF THE CENTRE 0<=Y<=3FH
; A REG = 0 TO UNPLOT THE CIRCLE, 1 TO PLOT THE CIRCLE

CIRCLE:     LD          (PLTFLG),A
            LD          A,H
            LD          (GCURX),A
            LD          A,L
            LD          (GCURY),A
            LD          D,0
            RST         18H
            DC          B(PRSET+4)

;************************
;ROUTINE TO TEST FOR CURSOR
;************************

;N.B. THIS SHOULD NOT BE CALLED BY USER, IT IS REQUIRED BY CHAR
;ROUTINE ONLY

REMCUR:     RST         18H
            DC          B(ERACUR)
```


Graphics Demonstration Program

The following CP/M program demonstrates how to use the graphics facilities described in this appendix. It displays text on the screen and then displays keyboard input to the screen. It is intended only to clarify the use of the routines.

The following directives to the Assembler may be unfamiliar to users of other assemblers. They should, however, have equivalent facilities; for details, consult your assembler manual:

DC	defines a constant
DC B(*addr*)	defines a double byte location to contain the 16 bit address *addr*. The least significant byte is stored first.
ORG	defines the actual address at which the following code is to be located.
EQU	direct equate of a symbol.

```
;************************************
;
;   DEMONSTRATION OF GRAPHICS PACKAGE
;
;************************************

; ram locations utilised

pltflg:     equ         0fa4fh
gcurx:      equ         0fa4dh
gcury:      equ         0fa4eh
addr:       equ         0fa53h
xpos:       equ         0fa51h
ypos:       equ         0fa52h
gchrflg:    equ         0fa55h
chrcell:    equ         0fa58h

; operating system dependant code

gcls:       equ         9G  08c25h ;9G06h  08c95h ;9G08h 08c95h
gmode:      equ             08bfch ;       08c66h ;      08c66h
tmode:      equ             08bbeh ;       08c28h ;      08c28h
ncuron:     equ             094a6h ;       094efh ;      094efh
ncuroff:    equ             0948bh ;       094d7h ;      094d7h
putcur:     equ             0908eh ;       09111h ;      09111h
curpxl:     equ             08c91h ;       08d01h ;      08d01h
prset:      equ             08d6dh ;       08dd1h ;      08dd1h
eracur:     equ             090d6h ;       09129h ;      09129h

            org         100h

xfer:       equ         0c800h          ;use c800h common area
bdos:       equ         5h

start:      ld          hl,cls          ;copy fuctions up to common ram
            ld          de,xfer
            ld          bc,60h
            ldir
```

```
        call    xfer+0          ;clear the screen
        call    xfer+3          ;call to graph
        call    xfer+12         ;turn cursor off
        ld      d,-1
        ld      e,3
        call    xfer+15         ;select inverse video char set 3
        ld      de,mesg1
        ld      c,9
        call    bdos            ;print hello on screen
        ld      d,0
        ld      e,0
        call    xfer+15         ;select normal char set 0
        ld      e,15
        ld      c,2
        call    bdos            ;output cursor addr prefix
        ld      e,56
        ld      c,2
        call    bdos            ;x = 56
        ld      e,20
        ld      c,2
        call    bdos            ;y = 20
        ld      de,mesg2
        ld      c,9
        call    5h              ;output how are you message
        ld      a,1
        ld      e,10
        ld      h,144
        ld      l,32
        call    xfer+64         ;draw circle on the screen
        ld      d,10
        ld      e,60
line:   push    de
        ld      a,1
        call    xfer+50
        pop     de
        inc     d
        ld      a,d
        cp      230
        jr      nz,line         ;continue to draw
        ld      b,16
loop:   call    delay
        djnz    loop            ;delay for while
        ld      de,mesg3
        ld      c,9
        call    bdos
```



```
nokey:  ld      e,-1            ;wait for key
        ld      c,6
        call    bdos
        and     a
        jr      z,nokey
; echo charcater to the screen
        ld      e,a
        ld      c,6
        call    bdos
        jr      nokey
delay:  ld      hl,0
delay1: dec     hl
        ld      a,h
        or      l
        jr      nz,delay1       ;delay routine
        ret
mesg1:  dc      'hello$'
mesg2:  dc      'How are you?$'
mesg3:  dc      'press a key...$'

; THE UTILITIES PROVIDED

cls:    rst     18h
        dc      b(gcls)
graph:  rst     18h
        dc      b(gmode)
text:   rst     18h
        dc      b(tmode)
curon:  rst     18h
        dc      b(ncuron)
curoff: rst     18h
        dc      b(ncuroff)
char:   ld      a,(gchrflg)
        cp      a
        jr      z,char1
        push    de
        call    xfer+80         ;SPECIAL CODE REMCUR
        pop     de
        ld      a,e
        ld      (gchrflg),a
        xor     a
        ld      (addr),a
        ld      (addr+1),a
        ld      (xpos),a
        ld      (ypos),a
char1:  ld      a,d
        ld      (chrcell+27),a
        rst     18h
        dc      b(putcur)
plot:   ld      (pltflg),a
        ld      a,d
        ld      (gcurx),a
        ld      a,e
        ld      (gcury),a
        rst     18h
        dc      b(curpxl)
circle: ld      (pltflg),a
        ld      a,h
```

```
            ld      (gcurx),a
            ld      a,l
            ld      (gcury),a
            ld      d,0
            rst     18h
            dc      b(prset+4)
remcur:     rst     18h
            dc      b(eracur)

            end
```


Reading a Pixel's Colour

The following machine code routine simulates the POINT (x,y) function in BASIC. The code is operating system independent and may be located in any convenient location available in the memory, not necessarily in common RAM.

The x parameter is passed in the H register and the y parameter in the L register

To reduce the code size, parameters are not checked on entry. The user program must ensure that the parameter constraints are within the specified limits prior to execution, in order for the code to execute as expected. These constraints are: 0< =x< =239 and 0< =y< =63

The POINT routine allows the "colour" of individual pixels, to be determined, a '1' indicating the pixel is black and '0' indicating the pixel is white.

This routine may also be beneficial to BASIC applications, as it is much faster than the equivalent BASIC function POINT(x,y).

```
; Machine code POINT function
; This code is independent of operating system versions.
;
; enter with x coordinate in H register
;            y coordinate in L register
;
; Note:     240 >= x >= 0  and  64 >= y >= 0

            org     0C000h
point:      push    hl              ;keep user coordinates
            ld      h,0h
            ld      a,30

            ld      d,h
            ld      e,l
            ld      hl,0
            and     a
            jr      z,axhl1
axhl2:      add     hl,de
            dec     a
            jr      nz,axhl2
axhl1:      pop     bc
            ld      a,b
            srl     a
            srl     a
            srl     a
            ld      e,a
            ld      d,0
            add     hl,de
            ld      a,b
            and     07h             ;hl = addr - a
            push    af
```

```
readbyt:    ld          b,0ah
            ld          c,1
            call        lcdreg          ;update lcd control
            inc         b
            ld          c,h
            call        lcdreg
            ld          b,0dh
            call        lcdbusy
            ld          a,b
            out         (21h),a
            call        lcdbusy
            in          a,(3eh)
            call        lcdbusy
            in          a,(3eh)
; now determine the bit status
            pop         bc
            inc         b
point1:     rr          a
            djnz        point1
            ld          a,0
            ret         nc
            inc         a
            ret

lcdreg:     call        lcdbusy
            ld          a,b
            out         (21h),a         ;high byte to lcd from b
            call        lcdbusy
            ld          a,c
            out         (20h),a         ;low byte to lcd from c
            ret

lcdbusy:    in          a,(21h)         ;get lcd status
            rlca
            jr          c,lcdbusy       ;until bit 7 is empty
            ret

            end
```


Appendix X IBM2780 Protocol Menu Program for 9G06

Introduction

This appendix lists a BASIC program that can be used to set up the optional IBM2780 protocol via a menu, for units with version 9G06 of the operating system. For details of how to use it, see 'Introduction to Communications', in Part 2.

NOTE: A program for 9G08 and later versions of the operating system is included in 'Introduction to Communications', in Part 2.

SETSYNC Listing

```
REM Title: SETSYNC.HBA
REM Utility program to set Bisync parameters.

10 CLS:KEYOFF:CUROFF:ONBREAKGOTO90
20 LOCATE11,0:PRINT"Bisync Parameters"
30 LOCATE11,0:PRINT"----------"
40 PRINT"1 Record Len... 6 Int STX......."
50 PRINT"2 Buffer Len... 7 Retrys........"
60 PRINT"3 Single Rec... 8 Spc Compress.."
70 PRINT"4 Rec Padding.. 9 Default Values"
80 PRINT"5 US or IRS.... 0 EXIT TO SYSTEM"
85 GOSUB900
90 GOSUB800:CURON:INCHR" Your Choice ";YC:CUROFF
100 IFYC<48ORYC>57THENBEEP:BEEP:GOTO90
110 YC=YC-48:ONYCGOTO200,250,300,350,400,450,500,550,600
120 A=CALL(0):REM RETURN TO DEMOS

200 REM SET RECORD LENGTH
210 GOSUB800:CURON:INPUT;"Enter Record Length"; RL: CUROFF
220 IFRL<100RRL>255THENGOSUB800:BEEP:PRINT"LENGTH OUT OF RANGE";:GOTO248
230 IFRL>=BLTHENGOSUB800:BEEP:PRINT"LENGTH GREATER THAN BLOCK LENGTH";:GOTO248
240 POKE63571,RL:LOCATE15,2:PRINTSTR$(RL);"";:GOTO90
248 FORI=0TO600:NEXTI:GOTO210

250 REM SET BUFFER LENGTH
260 GOSUB800:CURON:INPUT;"Enter Buffer Length";BL:CUROFF
270 IFBL<100RBL>512THENGOSUB800:BEEP:PRINT"LENGTH OUT OF RANGE";:GOTO298
280 IFBL<=RLTHENGOSUB800:BEEP:PRINT"LENGTH LESS THAN RECORD LENGTH";:GOTO298
290 POKE63572,BL-256*FIX(BL/256), FIX(BL/256)
295 LOCATE:15,3:PRINTSTR$(BL);"";:GOTO90
298 FORI=0TO600:NEXTI:GOTO260

300 REM SELECT SINGLE RECORD BUFFER
310 IFPEEK(63574)=0THEN POKE63574,255ELSEPOKE63574,0
320 LOCATE15,5:IFPEEK (63574)=0THEN PRINT"NO"ELSEPRINT"YES"
330 GOTO90

350 REM SELECT RECORD PADDING
360 IFPEEK (63575)=0THEN POKE63575,255ELSEPOKE63575,0
370 LOCATE15,5:IFPEEK(63575)=0THENPRINT"NO"ELSEPRINT"YES"
380 GOTO90
```

```
400 REM SELECT US OR IRS
410 IFPEEK(63576)=0THENPOKE63576,255ELESEPOKE63576,0
420 LOCATE15,6:IFPEEK(63576)=0THENPRINT"US"ELSEPRINT"IRS"
430 GOTO90

450 REM SELECT INTERMEDIATE SIX
460 IFPEEK (63577)=0THENPOKE63577,255ELSEPOKE63577,0
470 LOCATE36,2:IFPEEK(63577)=0THENPRINT"NO"ELSEPRINT"YES"
480 GOTO90

500 REM SELECT RETRYS
510 IFPEEK(63578)=0THENPOKE63578,255ELSEPOKE63578,0

520 LOCATE36,3:IFPEEK(63578)=0THENPRINT"10"ELSEPRINT"INF"
530 GOTO90

550 REM SELECT SPACE COMPRESSION
560 IFPEEK(63579)=0THENPOKE63579,255ELSEPOKE63579,0
570 LOCATE36,4:IFPEEK(63579)=0THENPRINT"NO"ELSEPRINT"YES"
580 GOTO90

600 REM SELECT DEFAULT VALUES
610 POKE63571,80,200,0,0,0,0,0,0
620 GOTO20

800 REM WIPE PUT BOTTOM LINE OF SCREEN
810 LOCATE0,7:PRINTSPACE$(39);:LOCATE0,7:RETURN

900 REM INITIALISE SCREEN VARIABLES
910 RL=PEEK(63571):BL=PEEK(63572)+256*PEEK(63573)
914 LOCATE15,2:PRINTSTR$(RL);916 LOCATE15,3:PRINTSTR$(BL);
920 LOCATE15,4:IFPEEK(63574)=0THENPRINT"NO"ELSEPRINT"YES"
930 LOCATE15,5:IFPEEK(63575)=0THENPRINT"NO"ELSEPRINT"YES"
940 LOCATE15,6:IFPEEK(63576)=0THENPRINT"US"ELSEPRINT"IRS"
950 LOCATE36,2:IFPEEK(63577)=0THENPRINT"NO"ELSEPRINT"YES"
960 LOCATE36,3:IFPEEK(63578)=0THENPRINT"10"ELSEPRINT"INF"
970 LOCATE36,4:IFPEEK(63579)=0THENPRINT"NO"ELSEPRINT"YES"
990 RETURN
```


Index

/DO command, 222
/DO command (DEMOS), 58

2780
  see 'Communications protocols'
2780 character set
  Appendix IV

3780
  see 'Communications protocols'

9600 baud communications, 139

ABS function (BASIC), 237
Accessories, 151
  barcode readers, 152
  battery charger, 154
  light pens, 152
ACK/NAK
  see 'Communications protocols'
Alarm clock feature, 233
Alkaline manganese batteries, 20
Alternate character sets
  Appendix VI
AND (BASIC operator), 175
ARG function (BASIC), 238
Arrow keys
  see 'Cursor keys'
ASC function (BASIC), 239
ASCII characters
  converting to EBCDIC
    Appendix IV
  table
    Appendix III
Assembly language, 391, 393
  CONT mode, 428
  CP/M system calls
    see 'System calls'
  cycle time, 396
  EPROMs
    see 'EPROMs'
  escape codes
    see 'Escape codes'
  graphics commands
    see 'Escape codes'
  graphics routines
    Appendix IX
  memory map, 393
  paging, 396
  power-off, 428
  power key scanning, 396
  RAM disk organisation, 397
  stack, 396
  system calls
    see 'System calls'
  timing instructions, 396
  TURNOFF routine, 428
ATN function (BASIC), 240
Automatic power-off, 21
Automatic timeout, 23

Backlight option, 18
Barcode readers, 152
BAS command (DEMOS), 37
BASIC
  alarm clock feature, 233
  chaining programs, 231
  editor, 185
  error messages, 234
  executing programs from RAM disk, 222
  file handling
    see 'File handling commands'
  graphics, 190
  interrupt keys, 188
  introduction, 171
  keywords, 236
    also see the individual keyword
  limits, 188
  machine code
    see 'BASIC and machine code'
  memory overhead (program storage), 189
  memory overhead (run-time), 189
  power-handling commands
    see 'power handling commands'
  precision, 188
  redefining the keyboard, 186
  selecting BASIC, 171
  selecting from DEMOS, 33
  soft keys, 187
  syntax
    see 'BASIC syntax'
  using the Oracle GT
    Appendix VI
  variable structure, 183
BASIC and machine code, 192
  calling routines, 192
  converting code to data, 198
  passing arrays, 194
  passing multiple variables, 193
  passing simple variables, 192
  passing strings, 194
  POKE.HBA, 198
  reserving RAM, 194
  systems calls, 194

BASIC compilers, 386
BASIC programming techniques
  array searching, 201
  arrays, 199
  help text, 203
  help text in graphics mode, 204
  screen, 201
  screen - cursor addressing, 202
  screen - fast updating, 202
BASIC programs
  running from DEMOS, 33
BASIC syntax, 173
  double precision variables, 181
  expressions, 173
  keyword types, 173
  logical expressions, 175
  numeric arrays, 181
  numeric expressions, 174
  numeric variables, 181
  operator precedence, 178
  operators, 173
  program lines, 180
  relational expressions, 175
  string arrays, 182
  string expressions, 174
  string variables, 182
  truth tables, 175
  variables, 181
Batteries, 20, 456
  alkaline-manganese, 20
  charger, 154
  conserving power, 205
  data life, 458
  installing, 20
  life, 456
  low battery warning, 21
  operating modes, 456
  power consumption example, 457
  power save mode, 22
  rechargeable, 155
  recharging, 22
  types, 20
Battery
  capacity, 456
BEEP command (BASIC), 241
BREAK key
  see 'BRK'
Breakout code, 11, 34
  reset command, 47
BRK key, 13
BS key, 12
BSC, 105

C compilers, 388
CALL command (BASIC), 242
Caring for Hunter 2, 149
Case sealing, 149
Chaining programs, 231
Character set
  ASCII
    Appendix III
    Appendix IV
  EBCDIC
    Appendix IV
Checksum
  file, 27, 36
CHAR command (BASIC), 243
CHAR command (machine code)
  Appendix IX
CHR$ function (BASIC), 244
CIRCLE command (BASIC), 245
CIRCLE command (machine code)
  Appendix IX
CLCK command (DEMOS), 38
CLEAR command (BASIC), 246
Clear to Send
  see 'Communications, CTS'
Clock access from applications
  Appendix VI
Clock corruption, 26
CLOSE command (BASIC), 247
CLS command (BASIC), 248
CLS command (machine code)
  Appendix IX
COM command (BASIC), 249
COM files, 35
Command interpretation
  DEMOS, 31
Communications
  and programming, 130
  BAUD RATE option, 88
  buffering, 83
  configuring, 86
  connecting to the port, 85
  connector pinouts, 85
  CTS option, 88
  DCD option, 89
  DSR option, 89
  DTR option, 89
  ECHO option, 90
  error messages, 147
  handshaking, 96
  introduction, 82
  LF option, 90
  NULL option, 91
  power control, 83
  protocols
    see 'Communications protocols'
  Protocols
    see 'Communications protocols'
  PRTCL (protocol) option, 91
  PTY (parity) option, 92
  RI (Ring Indicator), 96
  RTS option, 93
  RX T/O option, 94
  SERIG option, 94
  starting, 130
  TX T/O option, 95
  using communications
    see 'Using communications'
Communications protocols
  2780, 102
  2780 - ACK0, 107
  2780 - ACK1, 107
  2780 - and binary data, 108
  2780 - and terminal emulation, 126
  2780 - card formats, 117
  2780 - configuring the protocol, 120
  2780 - data formats, 116
  2780 - DEL, 108
  2780 - DEL EOT, 108
  2780 - ENQ, 107
  2780 - EOT, 107
  2780 - ETB, 106
  2780 - ETX, 107
  2780 - example sequences, 111
  2780 - IRS, 108
  2780 - ITB, 107
  2780 - NAK, 108
  2780 - RVI, 108
  2780 - SETSYNC program
    Appendix X
  2780 - SETSYNC8 program, 120
  2780 - STX, 106
  2780 - transparency, 108
  2780 - WACK, 108
  2780 error-checking, 105
  2780 link control characters, 106
  2780 synchronisation, 103
  3780
    see '2780'
  ACK/NAK, 99
  asynchronous, 97
  ETX/ACK, 98
  IBM 2780
    see '2780'
  job cards, 102
  NONE, 97
  synchronous, 102
  SYSTIME, 99
  XON/XOFF, 97
Compilers
  see 'Programming languages'
COMS command, 86
COMS command (DEMOS), 40
Configuring
  2780 protocol, 120
  2780 protocol for 9G06
    Appendix X
  communications, 86
CONT command (BASIC), 250
CONT command (DEMOS), 42
CONT mode
  leaving, 34
  selecting, 33
Continuous connection, 23
Continuous execution, 33
Continuous operation, 459
Control codes
  in DEMOS, 31
Control key, 12
Conventions, 4
Conversion tables
  ASCII to EBCDIC
    Appendix IV
  decimal to hexadecimal
    Appendix V
  EBCDIC to ASCII
    Appendix IV
  hexadecimal to decimal
    Appendix V
COS function (BASIC), 251
CP/M programming
  see 'Assembly language'
  see 'System calls'
CP/M programs
  running from DEMOS, 33
Crashes, 25
  clock corruption, 26
  file corruption, 27
  I/O corruption, 26
  keyboard lockout, 26, 27
  program corruption, 26, 27
  system lock-out, 27
Crossed cable, 82
CRT command (BASIC), 252
CTL key, 12
CUROFF command (BASIC), 253
CUROFF command (machine code)
  Appendix IX
CURON command (BASIC), 253
CURON command (machine code)
  Appendix IX
Cursor
  moving the cursor, 17
Cursor addressing, 15
Cursor keys, 12

DATA command (BASIC), 254
Data storage lifetime, 458

DATE$ function (BASIC), 255
DAY$ function (BASIC), 256
Dealing with problems
  see 'Crashes'
Decimal to hexadecimal
  Appendix V
DEFSEG command (BASIC), 257
DEL key, 12
Demonstration programs
  Appendix VI
DEMOS
  command types, 31
  commands, 37
  control codes, 31
  error messages, 59
  how commands are interpreted, 31
  introduction, 29
  prompt, 31
  running a CP/M program, 33
  running BASIC programs, 33
  selecting, 29
  selecting BASIC, 33
  typing commands, 31
Developing software, 385
DIM command (BASIC), 258
Dimensions
  Appendix I
DIR command (DEMOS), 43
Drawing
  see 'Graphics'
DTA files, 35
DTE, 82
Dustproofing
  Appendix II

EBCDIC, 105
EBCDIC characters
  Appendix IV
EDIT command (BASIC), 260
EDIT command (DEMOS), 44, 61
Edit file command, 44
Editing Hunter BASIC files, 61
Editing text files, 61
Editor
  arrow keys, 68
  caps lock key, 69
  character insert key, 70
  commands (summary), 67
  delete keys, 71
  delete line command, 75
  error messages, 80
  exit command, 72
  file size, 64
  filenames, 63
  find text command, 74
  go to file end, 73
  go to file start, 73
  go to line end, 76
  go to line start, 76
  keys, 65
  save file command, 78
  scroll page command, 77
  selecting, 61
  selecting from BASIC, 185
  skip word keys, 79
  tab key, 69
Electromagnetic compatibility
  Appendix II
Emergency breakout code
  see 'Breakout code'
END command (BASIC), 261
Enter key, 11
Environment, 25
  Appendix II
EOF function (BASIC), 262
EPROMs, 431
  extra DEMOS commands, 437
  routines, 438
  startup options, 436
  structure, 433
  uses, 432
EQV (BASIC operator), 175
ERA command (DEMOS), 45
Erase files command, 45
ERL function (BASIC), 263
ERR function (BASIC), 263
ERROR command (BASIC), 264
Error handling, 263
Error messages
  *** Communication Failure ***, 147
  ** Failed, 80
  ** Wait, 80
  Aborted, 59, 169
  Bad name, 59
  Cannot make file, 169
  Cannot open file, 169
  communications, 147
  DEMOS, 59
  Disk error, 59, 169
  Disk full, 59, 80
  EDIT, 80
  File R/O, 80
  File read only, 169
  Memory overflow, 59
  Memory overflow!, 81
  No file, 59, 169
  Syntax, 169
  Syntax!, 60
Error messsages
  BASIC, 234

ESC key, 13
Escape codes, 442
  audible output, 449
  backlight, 448
  Caps lock, 447
  CHAR, 446
  CIRCLE, 447
  CUROFF, 444
  CURON, 444
  INS key, 448
  LINE, 444
  LOCATE, 445
  POINT, 444
  PRESET, 446
  PSET, 446
  reset breakout sequence, 448
  SCREEN, 446
  set breakout sequence, 448
  viewing angle, 448
ETX/ACK
  see 'Communications protocols'
Example programs
  Appendix VI
Execute BASIC program from disk, 58
EXECUTE command (DEMOS), 46
Execute TPA command, 46
Executing programs from RAM disk, 222
EXP function (BASIC), 265
Extensions, 35
Extents, 397

File checksum, 27
File control block (FCB), 399
File corruption, 27
File handling commands, 210
  disk free space, 212
  end of file, 262
  file numbers, 210
  random access, 213
File Manager
  see DEMOS
  selecting, 7
File Manager screen, 29
File numbers, 210
File types
  System files, 9
FILES command (BASIC), 266
Filing
  * in filenames, 35
  ? in filenames, 35
  and the editor, 63
  checksum, 36
  extensions, 35
  file integrity checks, 36
  filenames, 35
  introduction, 35
  wildcards, 35
FIX function (BASIC), 267
FN key, 12
FOR command (BASIC), 268
FORMAT command (DEMOS), 47
Format RAM disk, 47
FRE function (BASIC), 270
Function keys
  KEYS, 7
  switch key set, 50

GENERATE command (DEMOS), 438
GOSUB command (BASIC), 271
GOTO command (BASIC), 272
Graphics
  coordinates system, 190
  from BASIC, 190
  machine code routines
    Appendix IX
Graphics commands
  assembly language
    see 'Escape codes'
Graphics mode, 14
GRAPH command (machine code)
  Appendix IX

HBA files, 35
HELP command (BASIC), 274
Help key, 11
Hexadecimal to decimal
  Appendix V
HLP key, 11
Humidity indicator, 150
Hunter 2
  accessories, 151
  caring for, 149
  case sealing, 149
  dustproofing
    Appendix II
  electromagnetic compatibility
    Appendix II
  humidity indicator, 150
  operating conditions, 149
  operating environment
    Appendix II
  shelf life
    Appendix II
  shock resistance
    Appendix II
  specification
    Appendix I
  storing, 150

waterproofing
Appendix II

I/O addresses, 421
I/O corruption, 26
IBM 2780
see 'Communications protocols'
Identifying the o/s version
Appendix VI
IF command (BASIC), 275
IMP (BASIC operator), 175
INCHR command (BASIC), 277
Increasing battery life, 205
INKEY$ function (BASIC), 279
INKEY function (BASIC), 278
INP command (DEMOS), 48
INP function (BASIC), 280
INPUT# command (BASIC), 286
INPUT command (BASIC), 281
Input file from RS232 port, 48
INPUT USING command (BASIC), 283
INS key, 13
Installing batteries, 20
INSTR function (BASIC), 287
INT function (BASIC), 288
Intel Hex
blocks
Appendix VII
file transfer example, 134
input command, 48
output command, 54
specification
Appendix VII
UNLOAD program
Appendix VI
utility
Appendix VI
Intel Hex conversion program
Appendix VI
Interfacing, 451
LEMO
see 'LEMO'
RS232 port
see 'RS232 port'
single bit input
see 'LEMO'
wand input
see 'LEMO'
Interpreters
see 'Programming languages'
Introduction
assembly language, 393
BASIC, 171
batteries, 20
communications, 82
DEMOS, 29
escape codes, 442
File Manager, 29
filing, 35
Hunter 2, 1
keyboard, 11
Oracle GT, 156
programming languages, 385
Screen, 14
the editor, 61
this manual, 3
troubleshooting, 24
using communications, 129

JSR$ function (BASIC), 289

KEY(n) command (BASIC), 292
Key codes, 423
KEY command (BASIC), 290
Keyboard
introduction, 11
layout, 10
redefining, 423
Keyboard lockout, 26, 27
KEYBUF, 423
Keys
BRK, 13
BS, 12
CTL, 12
Cursor, 12
DEL, 12
ENTER, 11
ESC, 13
FN, 12
HLP, 11
INS, 13
LBL, 13
PWR, 11
RETURN
see 'ENTER'
SHIFT, 12
terminal emulation, 137
KEYS, 7
KEYS command (DEMOS), 50
KILL command (BASIC), 294

Languages
see 'Programming languages'
LBL key, 13
LEFT$ function (BASIC), 295
LEMO, 454
electrical characteristics, 454
LEN function (BASIC), 296
LET command (BASIC), 297
Light pens, 152
LINCHR command (BASIC), 299
LINE command (BASIC), 300
Line operation, 155
LINPUT command (BASIC), 301
LIST command (BASIC), 302
List file contents command, 57
List file details command, 55
List user files, 43
LLIST command (BASIC), 303
LLOAD command (BASIC), 304
LN function (BASIC), 306
LOAD command (BASIC), 307
LOAD command (DEMOS), 51
Load file into TPA, 51
LOC function (BASIC), 308
LOCATE command (BASIC), 309
LOG function (BASIC), 310
LOPCHR command (BASIC), 311
Low battery warning, 21
Low power warning, 206
LPRINT command (BASIC), 312
LTRON command (BASIC), 313

Machine code
see 'Assembly language'
Mains operation, 155
Mains power supply, 461
Mains powered installation, 459
MAXFILES variable (BASIC), 314
MBASIC, 385
Memory locations, 414
BIOS jump table, 414
Memory map, 393
Memory paging, 396
Microsoft BASIC, 385
MID$ function (BASIC), 315
Modem eliminator, 82
Monitoring low power from an application
Appendix VI
Moving the cursor, 17

NAME command (BASIC), 316
NEW command (BASIC), 317
NEXT command (BASIC), 318
NOT (BASIC operator), 175

ON BREAK command (BASIC), 319
ON COM command (BASIC), 320
ON COMMS command (BASIC), 321
ON ERROR command (BASIC), 322
ON KEY command (BASIC), 325
ON n GOSUB command (BASIC), 323
ON n GOTO command (BASIC), 324
ON POWER command (BASIC), 326
ON POWER RESUME command (BASIC), 327
ON TIME$ command (BASIC), 328
OPCHR command (BASIC), 329
OPEN command (BASIC), 330
Operating environment, 25
Appendix II
Operating system addresses
see 'Memory locations'
Options
see 'Accessories'
OR (BASIC operator), 175
Oracle GT
baud rate selection, 159
charging, 158
disk commands, 165
DISKBOOT.HBA, 163
disks, 156
error messages, 169
HDOS.COM, 161
introduction, 156
LEDs, 157
loading DSK.COM, 161
operating, 156
power management, 157
power status, 157
power up, 156
using from BASIC
Appendix VI
using with Hunter 2, 165
OUT command (BASIC), 332

Paging, 396
Pascal, 387
PEEK function (BASIC), 333
Permanent charging, 459
PI constant (BASIC), 334
Pitch parameters
Appendix VIII
PLOT command (machine code)
Appendix IX
POINT command (machine code)
Appendix IX
POINT function (BASIC), 335
POKE.HBA, 198
POKE command (BASIC), 336
POP function (BASIC), 337
Ports, 421
POS function (BASIC), 338
Power
batteries
see 'Batteries'
continuous connection, 23
line powered, 459
mains powered, 459

permanent charging, 459
trickle charging, 459
Power-off, 21
Power-off timeout, 209
POWER command (BASIC), 339
Power consumption, 456
Power consumption example, 457
POWER CONT command (BASIC), 340
Power control
RS232 port, 130
Power handling commands, 208
disable power off, 208
handle power key, 209
power-off timeout, 209
power off with resume, 209
set power-off timeout, 208
turn power off, 208
Power modes, 205
POWER OFF command (BASIC), 341
POWER OFF RESUME command (BASIC), 342
Power save mode, 22, 205, 456
Power supply, 20
Precautions, 149
PRESET command (BASIC), 347
PRESET command (machine code)
Appendix IX (see PLOT)
PRINT# command (BASIC), 345
PRINT command (BASIC), 343
Problems
see 'Crashes'
Program corruption, 26, 27
Programming languages, 385
assemblers, 391
also see 'Assembly language'
C, 388
contact addresses, 391
cross compilers, 389
ordering, 392
other BASICs, 385, 386
Pascal, 387
Protocols
barcode
see 'Barcode'
communications
see 'Communications protocols'
PSET command (BASIC), 347
PSET command (machine code)
Appendix IX (see PLOT)
PUSH command (BASIC), 348
PWR key, 11

RAM disk, 35
RAM disk organisation, 397
Random access routines, 213
READ command (BASIC), 349
Rechargeable batteries, 155
Recharging batteries, 22
Reconfigure RS232 port command, 40
Redefining the keyboard, 423
REM command (BASIC), 350
REN command (DEMOS), 52
Rename file command, 52
Reprogram clock command, 38
Request to send
see 'Communications, RTS option'
RESTORE command (BASIC), 351
Restoring a deleted file
Appendix VI
RESUME command (BASIC), 352
RETURN command (BASIC), 353
RETURN key
see 'ENTER'
RIGHT$ function (BASIC), 354
RND function (BASIC), 355
ROM n command (DEMOS), 437
ROM VER n command (DEMOS), 438
RS232 port
electrical characteristics, 452
pinouts, 451
power control, 453
signals, 451
RTC access from applications
Appendix VI
RUN command (BASIC), 356
Run program in CONT mode, 42

SAVE command (BASIC), 357
SAVE command (DEMOS), 53
Save contents of TPA as file, 53
Screen
backlight, 18
coordinates, 15
cursor addressing, 15
introduction, 14
modes, 14
virtual screen, 15
SCREEN command (BASIC), 358
SCREEN command (machine code)
Appendix IX (see GRAPH, TEXT)
Sectors, 397
Select BASIC command, 37
Selecting
DEMOS, 29
the File Manager, 7
Selecting BASIC from DEMOS, 33
SEND command (DEMOS), 54
Send data to RS232 port, 54
SETSYNC program
Appendix X
SETSYNC8 program, 120
SGN function (BASIC), 359
Shock resistance
Appendix II
SHIFT key, 12
SIN function (BASIC), 360
Software development, 385
Software manufacturers, 391
SOUND command (BASIC), 361
parameters
Appendix VIII
SPACE$ function (BASIC), 362
SPC print function (BASIC), 363
Special key codes, 425
Specification
general
Appendix I
operating environment
Appendix II
SQR function (BASIC), 364
SRCH function (BASIC), 365
STAT command (DEMOS), 55
STOP command (BASIC), 367
Storing Hunter 2, 150
STR$ function (BASIC), 368
STRING$ function (BASIC), 369
SWAP command (BASIC), 370
Switching on, 6, 11
SYS files, 35
System calls, 398
close file (16), 405
compute file size (35), 409
console input (1), 401
console output (2), 401
delete file (19), 406
direct console I/O (6), 402
fetch key without echo (47), 411
file control block (FCB), 399
force transmission (50), 412
get allocation address (27), 407
get console status (11), 404
get disk parameter address (31), 408
get IOBYTE (7), 402
get read only vector (29), 407
get serial status (42), 410
get user code (32), 408
get version number (12), 404
hex format input (46), 411
hex format out (45), 411
initialise clock (39), 409
initialise communications (41), 410
list output (5), 401
make file (22), 406
open file (15), 405
print string (9), 402
punch output (4), 401
read clock (48), 412
read console buffer (10), 403
read random (33), 408
read sequential (20), 406
reader input (3), 401
rename file (23), 406
reset disk system (13), 404
reset drive (37), 409
return current disk (25), 407
return login vector (24), 407
search for first (17), 405
search for next (18), 405
select disk (14), 404
serial output status (54), 413
set DMA address (26), 407
set file attributes (30), 408
set IOBYTE (8), 402
set random record (36), 409
set up serial port (38), 409
set user code (32), 408
summary, 398
system reset (0), 401
terminal emulation (43), 410
transmit handshake, 412
wand input (44), 411
write protect disk (28), 407
write random (34), 408
write random with zero fill (40), 410
write sequential (21), 406
SYSTEM command (BASIC), 371
System files, 9
System lock-out, 27
SYSTIME
see 'Communications protocols'

TAB print function (BASIC), 372
TAN function (BASIC), 373
TERM command, 137
TERM command (DEMOS), 56
Terminal emulation, 137
selecting, 137
with 2780 protocol, 126
Terminal emulation command, 56
TEXT command (machine code)
Appendix IX
Text mode, 14
TIME$ function (BASIC), 374
Timeout
Hunter idle, 23
power-off, 209
receive (communications), 94
transmit (communications), 95
TPA, 393
Trickle charging, 459

TROFF command (BASIC), 375
TRON command (BASIC), 375
Troubleshooting, 24
Turning on, 6
Turning on and off, 11
TURNOFF routine, 428
TXT files, 35
TYPE command (DEMOS), 57
Typefaces, 4

UNLOAD program
Appendix VI
Upgrades
see 'Accessories'
USASCII
Appendix III
USER address, 414
Using alternate character sets
Appendix VI
Using communications, 129
advanced techniques, 139
advanced techniques - 9600 baud, 139
advanced techniques - using bisynch, 141
advanced techniques - using SYSTIME, 145
error messages, 147
file transfer, 131
file transfer (binary files), 132
file transfer (Intel Hex format), 134
from programs, 130
standard applications, 129
terminal emulation, 137
Using Hunter 2, 149
Using the RTC from applications
Appendix VI

VAL function (BASIC), 376
VARPTR function (BASIC), 377
Vectors
see 'Memory locations'
Virtual screen, 15
automatic window movement, 16
moving the window, 15

Wand input port
see 'LEMO'
WAND variable (BASIC), 378
Waterproofing
Appendix II
Weight
Appendix I
Welcome Message, 29
WEND command (BASIC), 379
WHILE command (BASIC), 379
Wildcards in file names, 35
WINCHR command (BASIC), 381
WINPUT command (BASIC), 382
WRITE# command (BASIC), 383

XON
see 'Communications protocols'
XOR (BASIC operator), 175

Z80 programming
see 'Assembly language'

Notes:

Notes:

Notes:

Notes:

Notes:

Notes:

Notes

Notes:

Husky Hunter 2 User's Guide

Notes:

Notes:

Notes:

Notes:

Notes:

Notes:

Notes:

Notes

Notes:

Husky Hunter 2 User's Guide

Notes:

Notes

Notes:

Notes:

Notes:

Notes:

HUSKY™

Husky Computers Limited, P.O. Box 135 345 Foleshill Road, Coventry CV6 5RW England.
Telephone (0203) 668181 Telex 317450 HUSKY G Fax (0203) 680612